Analysis and Synthesis of Interactive Video Sprites

Corneliu Ilisescu

Department of Computer Science

University College London

This thesis is submitted for the degree of

Doctor of Philosophy

Declaration

I, Corneliu Ilisescu, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Corneliu Ilisescu
January 2018

Abstract

In this thesis, we explore how video, an extremely compelling medium that is traditionally consumed passively, can be transformed into *interactive experiences* and what is preventing content creators from using it for this purpose.

Film captures extremely rich and dynamic information but, due to the sheer amount of data and the drastic change in content appearance over time, it is problematic to work with. Content creators are willing to invest time and effort to design and capture video so why not for manipulating and interacting with it? We hypothesize that people *can help* and *be helped by* automatic video processing and synthesis algorithms when they are given the right tools.

Computer games are a very popular interactive media where players engage with dynamic content in compelling and intuitive ways. The first contribution of this thesis is an in-depth exploration of the *modes of interaction* that enable game-like video experiences. Through active discussions with game developers, we identify both how to assist content creators and how their creation can be dynamically interacted with by players. We present concepts, explore algorithms and design tools that together enable *interactive video experiences*.

Our findings concerning processing videos and interacting with filmed content come together in this thesis' second major contribution. We present a *new medium of expression* where video elements can be looped, merged and triggered interactively. Static-camera videos are converted into loopable sequences that can be controlled in real time in response to simple end-user requests. We present novel algorithms and interactive tools that enable our new medium of expression. Our human-in-the-loop system gives the user progressively more creative control over the video content as they invest more effort and artists help us evaluate it.

Monocular, static-camera videos are a good fit for looping algorithms but they have been limited to two-dimensional applications as pixels are reshuffled in space and time on the image plane. The final contribution of this thesis breaks through this barrier by allowing users to interact with filmed objects in a three-dimensional manner. Our novel object tracking algorithm extends existing 2D bounding box trackers with 3D information, such as a well-fitting bounding volume, which in turn enables a new breed of interactive video experiences. The filmed content becomes a *three-dimensional playground* as users are free to move the virtual camera or the tracked objects and see them from novel viewpoints.

Acknowledgements

First of all, I would like to thank my advisor, Dr. Gabriel Brostow, for his invaluable advice, positive attitude and guidance throughout my studies. He has always inspired me and shown me the right way. I would also like to thank my second supervisor, Dr. Neill Campbell, for his fresh ideas and endless enthusiasm.

I am very grateful to my collaborators, Aytac and Matteo, for being there when they were needed most. Thank you to CR-Play for funding this PhD and to all the people involved with the project for the great insight and great attitude. Thank you to all past and present members of the Prism group at UCL, as well as all the 5th floor officemates, for the endless discussions over lunch and for just being a pleasure to be around.

I would like to thank my family and friends. My parents and sister for supporting me and always being there especially in the darkest of times. Peter for his undeniable genius, hours upon hours of brainstorming and the late night discussions about anything and everything. Moos and Clement for being the greatest of friends and pushing me to be my best self. Lucy, Tracy, Clement and Carolina for being great people and always making me feel happier. Charlie, Frank, Dennis, Dee and Mac for bringing light into my life in the hard last months.

Last but definitely not least, Tara. You are the best thing that happened to me. Words cannot describe how much you mean to me. This is for you.

Contents

Contents viii

List of Figures xii

List of Tables xiv

1 Introduction 2

1.1 Research questions . . . 3

1.2 Brief Overview . . . 4

1.3 Contributions of this thesis . . . 6

2 Literature Review 8

2.1 Video Looping . . . 8

2.2 Video Editing and Authoring . . . 11

2.2.1 Video-based animation . . . 15

2.3 Multi-view from Single view . . . 16

2.3.1 Image-based Rendering . . . 20

2.4 Conclusions . . . 22

3 Problem Analysis 24

3.1 Indefinite Video Playback . . . 27

3.1.1 Objective Distances . . . 29

3.1.2 Perceptual Distances . . . 32

3.1.3 Considerations . . . 36

3.2 Controlling Video Output . . . 37

3.2.1 Semantic Looping . . . 38

3.2.2 Speed Normalization . . . 42
3.2.3 Considerations . . . 46
3.3 Real-time Interaction . . . 47
3.3.1 Video Fields . . . 47
3.3.2 Considerations . . . 52
3.4 Foreground Segmentation . . . 53
3.4.1 Example-based Segmentation . . . 54
3.4.2 Intensity-based Segmentation . . . 55
3.4.3 Considerations . . . 56
3.5 Video Authoring . . . 57
3.5.1 Creating Video Textures for Video Games . . . 58
3.5.2 Considerations . . . 61
3.6 Multiview Interaction . . . 62
3.6.1 Generating 3D Visuals from 2D Video . . . 63
3.6.2 Considerations . . . 66
3.7 Conclusions . . . 66

4 Responsive Action-based Video Synthesis 68
4.1 System Overview . . . 70
4.2 Actor Preparation . . . 73
4.2.1 Tracking . . . 73
4.2.2 Segmentation . . . 74
4.2.3 Action Definition . . . 75
4.3 Video Performance . . . 78
4.3.1 Frame Compatibility . . . 79
4.3.2 Action-based Video Synthesis . . . 81
4.4 Practicalities . . . 82
4.4.1 Real-time Performance . . . 83
4.4.2 Optimization Compression . . . 84
4.4.3 Post-Processing Rendering . . . 84
4.4.4 Precomputing Loops . . . 84
4.5 Creative Synthesis . . . 85
4.6 Results . . . 87

4.6.1 Counter Loop 89
4.7 Empowerment Evaluation 90
4.8 Discussions with Artists 92
4.9 Conclusions 93

5 Multi-view from single-view **96**
5.1 Camera Estimation 97
5.1.1 User-in-the-loop estimation 101
5.2 Well-grounded Tracking 103
5.2.1 Multiple objects 2D Tracks 104
5.2.2 Estimating and Tracking Cuboids on the Ground 104
5.3 Applications 110
5.3.1 From Tracked Cuboids to Textured Models 110
5.3.2 Video-Based Rendering 114
5.4 Results 116
5.4.1 Tracking 117
5.4.2 Interactive 3D Video Experiences 120
5.5 Conclusions 122
5.5.1 Limitations and future work 123

6 Conclusion **126**
6.1 Possible Future Directions 128

References **131**

List of Figures

3.1 Video-based game asset of a candle flame . . . 25
3.2 Distance metric differences . . . 28
3.3 Foreground/background grid encoding . . . 31
3.4 HOG features . . . 33
3.5 Weighted L2 Distance . . . 36
3.6 Label propagation results . . . 37
3.7 RIBBON dataset . . . 39
3.8 Label Propagation Training . . . 41
3.9 Regularly sampling a 2D trajectory . . . 43
3.10 Speed normalization software . . . 45
3.11 The effect of γ on video looping . . . 50
3.12 The effect of γ on video field traversal . . . 52
3.13 Example-based segmentation UI . . . 55
3.14 Intensity-based segmentation UI . . . 56
3.15 Video texture creation UI . . . 58
3.16 Pendulum dynamics . . . 59
3.17 Flag video texture in game levels . . . 62
3.18 Generating 3D Visuals from 2D Video . . . 64

4.1 Responsive Action-based Video Synthesis concept pipeline . . . 69
4.2 Overview of our interactive video synthesis pipeline . . . 71
4.3 Actor preparation user interface . . . 73
4.4 User-in-the-loop segmentation and compositing . . . 74
4.5 Action definition through label propagation . . . 76
4.6 Label propagation *vs.* manual annotations . . . 77

4.7 Video synthesis user interface 78
4.8 Frame compatibility illustration 80
4.9 Visual representation of the energy terms in Equation 4.5 82
4.10 Precomputed control graph 85
4.11 Creative action triggers 86
4.12 Sample frames from our output videos 87
4.13 Our video game prototype: Counter Loop 90
4.14 Video synthesis user study timings 92

5.1 Line segments 99
5.2 Camera pose estimation based on horizon line and ground plane . 101
5.3 Tool for manually estimating camera pose and scene geometry . . 102
5.4 Spline-based trajectory smoothing 106
5.5 Well-grounded tracking energy terms 108
5.6 3D reconstruction cost volume 112
5.7 Free-viewpoint interactive video 115
5.8 Manual vs automatic bounding volume tracking 119
5.9 Reconstruction and 3D VBR results 121
5.10 3D Reconstruction and VBR limitations 123

List of Tables

4.1 Our Responsive Action-based Video synthesis datasets 88

5.1 Tracking and 3D reconstruction datasets 116
5.2 Tracking accuracy of our automatic algorithm compared to manually tracked objects in a variety of input videos 118

LIST OF TABLES

Chapter 1

Introduction

Films and video games are two extremely appreciated entertainment media and two of the biggest earning businesses worldwide, generating billions yearly. While both are enjoyed by millions of people, they are inherently different. Films are essentially consumed passively while video games are *highly interactive* and users play an active role in what they are presented with. In recent years, big video game companies have invested much effort into producing more refined film-like experiences where, along with more complex and involving stories, *visual fidelity* has played a very important role.

Photo-realism has long been a central goal to the video game industry. While big leaps have been made towards achieving it, it still eludes us. Additionally, massive amounts of resources and hundreds of talented people are required to create ever more visually striking and technically complex video games. Recently, a new breed of algorithms have become available to game developers. Never before seen levels of realism are achieved with less effort by directly replicating the real-world appearance of single objects or full environments captured through photography. On the one hand, photogrammetry methods such as [FG14] allow people to capture the appearance of objects and use them as traditional video game assets by simply taking photos of them from multiple viewpoints. On the other hand, image-based rendering (IBR) methods such as [LH96, BBM*01] and more recently [CDSHD13, HRDB16, HASK17] can synthesize new views of a photographed scene by interpolating between captured ones and thus allowing users to move freely in a photo-realistic 3D environment.

While incredibly successful, all the methods above make the *critical* assumption that the scenes or objects captured in the input photos are static. As a consequence, one is presented with a completely static environment. While it looks photo-realistic, it also feels fake as the movement present in everyday life, such as the subtle movement of trees in the wind, clouds in the sky or even passersby, are missing from the experience. Moreover, there are no easy ways to manipulate IBR or photogrammetry assets as shape and appearance are baked-in to replicate the captured images. In contrast to photos, videos can capture dynamic elements and effects over time, so it stands to reason that a realistic experience would benefit from using this type of media.

In this thesis, we explore how dynamic elements that can only be captured with a video camera, can be re-introduced into otherwise static experiences such as those produced by traditional IBR methods or even into video game engines to be used alongside traditional and image-based assets. We are also concerned with devising techniques that give users the ability to manipulate and be creative when making video assets. In parallel, we investigate how videos can be used for interactive experiences, whether standalone or from within a video game setting. These are aimed at actively engaging end-users as opposed to treating them as passive consumers of an immutable medium such as film.

1.1 Research questions

As mentioned above, stills have long been used for photo-realistic interactive content (*e.g.* photogrammetry [FG14]) but they are inherently static. In contrast, videos capture dynamic events that happen over time but, as such, are harder to work with. In this thesis, we hypothesize that *interactive tools and automation can assist content creators in the creation of video experiences to engage audiences in new ways.*

We test this hypothesis mainly by addressing the following research question: in what ways can *watching videos become a more interactive experience* if content creators are actively involved in the process of reasoning about and synthesizing videos. Practically speaking, we are interested in going beyond simple infinite playback (*e.g.* [SSSE00]) and identifying meaningful ways in which *videos can*

react to human action. Moreover, we would like to define the type of information that needs to be inferred from a simple sequence of stills to enable real-time interaction between humans and video content. Finally, we are concerned with devising techniques that support content creators to ensure efficient processing and building of interactive video experiences.

1.2 Brief Overview

We now give a brief overview of how this thesis is organized. In Chapter 3, we take a close look at the data we capture when recording dynamic elements over time. In particular we seek a better understanding of what are desirable qualities of interactive videos and how they can be accomplished. We find that there are two main limitations inherent to traditional film experiences: a) their limited duration and b) the lack of information to leverage for meaningful real-time interaction.

In an interactive setting, such as video games, players are free to interact with and enjoy the experience for as long as they like. Since we cannot capture and store an infinite amount of video frames or even efficiently process large quantities of images, we must find ways to give the impression the video is playing indefinitely. In 2000, Schödl *et al*. introduced the Video Texture [SSSE00], a seemingly endless stream of images created from a finite length video. Videos are played back indefinitely *without visible cuts*, such as those seen when a video reaches its end and starts over, by jumping around the original time-line wherever seamless transitions are possible. This concept is commonly know as *video looping* and many researched it extensively over the years such as [SSSE00, KSE*03, LJH13, SLWSS15] to name just a few. The core idea behind all of these methods is to find visually similar pairs of frames or patches that can be used interchangeably. Finding said pairs becomes quickly troublesome when we widen the range of videos we tackle beyond those that are easy to loop. Simply comparing the appearance of two frames by, for instance, taking the difference between their pixels' intensity, can prove inaccurate. We therefore explore alternatives in Section 3.1.

While an indefinitely looping video can be entertaining on its own, we believe interactivity is crucial to immersing audiences into a video experience, giving it a more game-like appeal. This could take the shape of a fluttering flag correctly

reacting to the player changing the direction or intensity of the wind. Or it could look like a street crossing where players are given the ability to select which cars should move and how. Or it could even mean giving users the power to control which drums a drummer is seen hitting in a video by simply pressing buttons on a keyboard. In Sections 3.2 and 3.3 we explore ways of introducing additional information into the process of creating new video content and how to make this reactive to users in real time. The remainder of Chapter 3 explores additional research problems we believe are important to making interactive videos a reality, such as effective authoring tools for content creators (Sec. 3.5) and multi-view capabilities similar to IBR (Sec. 3.6).

In Chapter 4 we take some of the ideas discussed in our problem analysis chapter, improve upon and package them in an end-to-end software package that enables the creation of interactive video experiences. Informed by numerous discussions with game developers that were actively involved in the development of some of the ideas presented in this thesis, we make the conscious choice to cater for both content creators and content consumers. As content creators, game developers generally expect better results if they invest more effort. Therefore, our system is designed to enable users' active involvement in helping and being helped by otherwise automatic algorithms throughout the "interactive videos creation pipeline". We also informally explore and discuss the human-computer interaction (HCI) side of the coin. This includes investigating how our system helps content creators but also how casual users can create new videos in real time with just a few button presses.

Finally, we dedicate Chapter 5 to the next step towards turning traditional videos into a game-like interactive medium. As we discuss in Section 3.6, we would like to reason about videos in terms of a 3D world, as opposed to the purely 2-dimensional images discussed in Chapter 4. This reasoning brings us closer to modern day video game scenarios. We show in Section 4.6.1 how we can let players decide when cars should move through a street crossing but what if we wanted to let them change *how* the car moves? We would have to know how the car looks like when it's seen from a different viewpoint, how it moves in the real world and how it changes appearance based on how far away from the camera it is. We make strides towards recovering this type of information by

devising a new method for tracking 3-dimensional objects given 2-dimensional boxes tracked over time in the input video. This informs us about how the object looks like from different viewpoints and how it gets occluded by and occludes other objects present in the filmed scene. It can also help automate some of the painfully manual processes described in Chapter 4 such as tracking moving objects and separating them from the background, thus reducing necessary user effort while keeping quality high. Finally, we show how it enables novel applications such as 3D reconstruction (Section 5.3.1) and revealing more information through novel view synthesis (Section 5.3.2).

1.3 Contributions of this thesis

In this thesis we have made strides towards a better understanding of how video data can be used in interactive and real-time settings. Our contributions are as follows:

- an in-depth exploration of the problems surrounding the processing and packaging of video data for interactive use cases, along with an investigation of potential research directions;
- a new medium of expression, akin to audio live looping, that allows one to create new videos in real-time by simply requesting filmed subjects to perform user-defined visually distinct actions;
- an end-to-end system that enables the above mentioned medium of expression by providing content creators the tools to process, annotate and synthesize video content;
- a novel object tracking algorithm that extends existing 2D bounding box trackers by estimating a 3D bounding volume and placing it on the 3D world-space ground plane which enables new applications such as reconstructing the three-dimensional shape of the tracked objects.

Publications

Parts of this thesis have been published in the following publication:

Ilisescu, C., Kanaci, H. A., Romagnoli, M., Campbell, N. D., Brostow, G. J.: Responsive Action-based Video Synthesis. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, (2017), ACM, pp. 6569-6580.

Chapter 2

Literature Review

In this chapter, we present the related literature to the topics central to this thesis. Each section tackles a different topic and several inspirational works and current state of the art approaches are discussed. In Section 2.1 we discuss *video looping*, the difficulties it presents and proposed solutions. Section 2.2 is concerned with *authoring video* content, algorithms for synthesizing and editing videos and the interactive tools they are part of. Issues related to creating *multi-view* outputs from monocular static or moving cameras are presented in Section 2.3 and to briefly exploring the *image- and video- based rendering* literature and how they enable interactive experiences.

2.1 Video Looping

Looping a video refers to the ability to play back a finite-length video for an indefinite amount of time without visible cuts or transitions. When a video reaches its end, playing it back from the start results in a disturbing jitter effect as a clear jump or transition occurs. This is due to the fact that images at the beginning and the end of a filmed sequence do not typically look alike. The key technique that makes video looping possible is the ability to find frames within a video corpus that have a similar appearance.

The pioneering work by Schödl *et al.* [SSSE00] introduced the concept of *video texture*: a middle-ground between image and video where dynamic events

happen indefinitely without visible cuts. It worked by finding interchangeable pairs of frames and using them to jump across the original time-line seamlessly. Given the similarity between every possible pair of frames, which was defined as the Euclidean distance between each pixel's color intensity, Schödl *et al.* pre-compute finite-length seamless loops. They could be combined to create a longer, more varied looping video using dynamic programming. Similar work had been previously presented by Bregler *et al.* in *Video Rewrite* [BCS97] where video sequences of people speaking were re-composited to match a given arbitrary audio track. Unfortunately, assuming that similar pairs of frames exist limits the types of videos drastically to ones showing a single subject or depicting repetitive or stochastic motion. In fact, such methods struggle with videos containing multiple independent subjects or complex motions.

In order to cater for a wider range of videos that may contain large or non-repetitive motions and, more generally, large changes in appearance over time, some have attempted to treat groups of pixels separately from one another. For instance, one could group pixels based on whether they contain the same object over time or whether they show similar motion. These patches are commonly know as *video sprites* and are often used in the literature to break a sequence of images into contained regions of similar appearance or behavior. This term was originally introduced by Pollard *et al.* to represent an alpha matted region of a video in their work about view interpolation [PPHL98]. However, a similar concept, called *video clip-art*, was used by Finkelstein *et al.* to allow alpha-matted video regions to be freely composited together in their spatial and temporal multi-resolution work [FJS96].

Generally speaking, the process of defining temporally-coherent patches corresponds to segmentation as evidenced by the work on texture synthesis by Kwatra *et al.* [KSE*03]. Their *Graphcut Textures* work enables the creation of larger pictures or looping videos by shuffling in space and time patches of pixels using an energy minimization approach. A Markov Random Field (MRF) is defined over the pixel grid, and in the case of video over the three-dimensional pixel volume, and the best matching seams are found using graph cuts [BVZ99]. Extensions for panoramic videos [AZP*05] and stereo panoramic videos [CLR11] where also explored. In [AZP*05], Agarwala *et al.* first find video loops in dynamic regions

of the input video which are manually defined by users. They then merge looped dynamic regions with static regions using a two step process that uses dynamic programming to prune the search space for a full MRF. In contrast, Couture *et al.* [CLR11] choose to use full frames and focus on the additional registration issues introduced by their stereo input videos. They do not need a graph cut-based solution and show they can reach good results by simply blending looped video blocks that overlap in time.

Similar to video textures, cinemagraphs [BB12] are still images that contain minor repetitive motions. While traditionally they were manually created using image and video editing software, a number of automatic and semi-automatic methods to simplify this tedious process have been devised in recent years. Often, methods derived from [SSSE00] are used to loop spatio-temporal patches of pixels. In [TPSK11], Tompkin *et al.* find motion in a stabilized input video automatically and allow users to manually select which regions to loop. They interpolate frames around visible jumps using bi-directional SIFT flow [LYT*08]. Joshi *et al.* [JMD*12] present a similar system but give users more control over the final video composition and use feathering to disguise temporal discontinuities. Bai *et al.* [BAAR12] use graph cuts to define video segments dynamically, loop them individually or de-animate them using warps, while in their follow-up work [BAAR13] focus specifically on portraits. Finally, Sevilla-Lara *et al.* [SLWSS15] focus on videos exhibiting camera motion which significantly increases the looping complexity and thus limit themselves to single dominant subjects.

Liao *et al.* present a more generic system in [LJH13, LFH15] where cinemagraphs are created by automatically finding video loops using a 2D MRF over the pixel grid. Their energy function is designed to ensure that pixel neighborhoods are spatially and temporally consistent while static pixels remain so if they are in a static region of the video. The goal of the optimization is to find, for each pixel in the MRF, a starting time s_x and a time period p_x over which the looping happens. Moreover, each pixel is assigned an activation threshold $a_x \in [0, 1]$ that indicates at which level of dynamism the pixel switches from being static to looping. The per-pixel activation thresholds and time periods define a segmentation of the video scene based on level of dynamism. Neighboring pixels are assigned to the same dynamic region if they have the same looping period and activation level and their

temporal extent overlaps. Users can interactively scribble over the segmented video to decide which areas to loop in the result.

All the techniques suffer from a number of limitations. First of all, all but [SLWSS15] assume the camera is static. Moreover, they assume there is only one [SSSE00, BAAR12, SLWSS15] or few [TPSK11, JMD*12] non-overlapping filmed objects. Even when no assumptions are made about the number of moving subjects [KSE*03, LJH13], they struggle with large motions such as when objects move across the video frame and even come in and go out of sight over time. Crucially, none of the presented systems give users the opportunity to correct mistakes such as when the looping or jump disguising algorithms struggle. Moreover, existing methods give little to no control over the synthesis process and only focus on finding jumps or disguising them. Users are usually limited to selecting patches of pixels as seen in [TPSK11, BAAR12, LJH13] or very specialized control such as controlling where a fish goes in a tank as shown in [SSSE00].

2.2 Video Editing and Authoring

Video looping is a powerful tool and allows new and interesting outcomes to be synthesized. However, most of the methods described in Section 2.1 can sometimes produce unexpected results and users have little to no way of intervening and correcting or controlling the synthesis process.

The first attempt at reintroducing user control over video synthesis comes from the original Video Textures paper by Schödl *et al.* [SSSE00]. In this paper, they allow videos to be used for animation by selecting frames based on other criteria other than whether they create a good looking loop. One such criteria is the range of frames to loop over which allows users to show different parts of a video such as a runner running at different speeds on a treadmill. In contrast, Bhat *et al.* [BSHK04] propose using particles to represent and synthesize phenomena such as waterfalls, smoke and fire. Users are required to draw lines (called flow lines) on the input video over the moving parts that they want to edit (such as water). Particles are then simulated along these lines and appearance at each time step is learned. To create a new video, the flow lines are arbitrarily rearranged and modified and are used to simulate new particles which are in turn used to render

the appearance of the moving phenomena. Examples of edited waterfalls and smoke are shown in their paper.

A number of techniques and associated tools have been developed to enable users to easily create cinemagraphs [TPSK11, BAAR12, JMD*12, LJH13]. In [TPSK11], Tompkin *et al.* allow users to click on regions they have automatically segmented based on amount of motion to decide whether they should be frozen in time or show an animation. In contrast, Joshi *et al.* [JMD*12] present a simple and intuitive non-linear video editor where users can quickly segment moving elements and animate them using one of three idioms: loop, mirror and playback. Many layers of moving and static elements can be composited together by using the provided user interface. Liao *et al.* [LJH13, LFH15] allows users to control which parts of the final video to loop by scribbling over the automatically found clusters of pixels. Bai *et al.* [BAAR12] allow users to guide their video synthesis algorithm by scribbling over pixels they want to control. Each scribble brush indicates to the system whether areas should be synthesized static, de-animated or looped.

The techniques described above are very powerful but present two major limitations. First, they only allow very specialized or limited control. For instance [BSHK04] assumes motion can be described by a particle simulation, [SSSE00] assume motion variations are filmed sequentially while the remaining only give users a binary choice of "(not) animate". Second, all the presented methods but [SSSE00] do not allow for real-time interaction. Videos are processed and edited using the provided tools but, once the choices of what and how to loop are made, they are recorded and rendered into a final video which can be simply enjoyed as traditional content.

A separate research thread in video editing methods is called *re-timing* and consists in reordering filmed events for new and interesting effects. Pritch *et al.* [PRAP08] focus on condensing large amounts of video into short animations. They track moving objects such as cars at a road crossing and composite them together to minimize the length of the final video while avoiding collisions. Klein *et al.* [KSC*01] represent videos as a spatio-temporal volume and allow artists to semi-automatically slice it to produce interesting artistic effects such as cubism. Similarly, Lu *et al.* [LZW*13] track moving objects to produce bendy tubes of

pixels in the video volume. Users are given the tools to manipulate these so-called *video sprites* by cutting, stretching and repositioning them in order to rearrange the original objects in the input video. Similarly, Shah and Narayanan [SN13] allow users to navigate the input video by scrubbing a cursor over an object's trajectory in the video volume and manipulate them by reordering, re-timing, cloning, removing or reversing them. Their system also deals with moving cameras, for which they can create wide field of view videos or synopses. *DuctTake* [RWSG13] performs spatio-temporal video compositing by finding seams in the video volume in a similar manner to [KSE*03]. Events filmed in the same scene but at different times can be composited together by scribbling over the input video frames. Liao *et al.* [LYGC15] build on this by allowing music to drive event re-ordering. Finally, Rav-Acha *et al.* [RAPLP05] bend time for image patches by projecting their pixels onto evolving time fronts.

Once again, the methods presented above suffer from the crucial limitation that the interactions with the content creator happen off-line, so they are not suitable to game-like scenarios. Often they also require a non-indifferent amount of user effort such as in the case of [LZW*13] where the manipulation of the bendy tubes mentioned above, while intuitive, require expertise and careful planning to avoid undesirable effects such as colliding cars. Moreover, the techniques above are limited to interactions with moving objects (*e.g.* cars) and the frame re-arrangement is only able to manipulate the time at which events happen. One could not, for instance, change the way a car moves.

Goldman *et al.* [GGC*08] argue that many interesting video editing applications can be made possible if we provide object tracking, annotations and compositions. Their system allows users to select objects in a video and directly control them by manipulating sets of tracked feature points. A sparse set of points are tracked over time and grouped together based on motion similarity. Users can then select subsets of them and use the mouse cursors to drag and drop points which in turn traverses the video timeline to show the associated motion. In [CPW*11], Chen *et al.* go the extra mile by providing intuitive tools for image-based modeling. They allow users to manually specify rough geometric shapes which in turn are used to constrain their 2.5-dimensional video representation, the "Video Mesh". Moving elements can be copied and placed consistently in the

scene and special effects such as smoke and depth of field can be added by using the depth information. Other systems focus on smart ways of navigating video content such as the Direct Manipulation Player by Dragicevic *et al.* [DRB*08] where users can drag filmed objects in the image plane as their point trajectories are associated to traversal of the video time-line. In [NNL14], Nguyen *et al.* allow similar functionality on mobile devices while in [NNL13], they allow users to intuitively navigate a video by dragging a cursor over 2-dimensional trajectories visualized in a 3-dimensional space-time volume.

The importance of tracking and segmentation

As previously mentioned, complex videos showing multiple objects moving independently, do not lend themselves to video looping techniques that use full frame statistics such as the original video textures paper [SSSE00]. As seen above, it is common to track objects or sets of pixels over time and segment them from the background. This simplifies the problem by dividing the input footage into multiple moving elements that can be treated separately. There are many methods to tackle this problem found in the literature.

Lu *et al.* [LZW*13] adopt a user-centric approach by allowing them to define ellipsoids encompassing the filmed objects at key frames and interpolating between them. The objects are then segmented based on their difference to the background. In [GGC*08], Goldman *et al.* track a large number of image points using [ST06] and grouping them into clusters based on motion similarity using a K-affines motion model. The moving objects in the final videos can then be composited using graph cuts. In contrast, Dragicevic *et al.* [DRB*08] use a feature-based optical flow estimation scheme to estimate object motion. A different approach is taken by Chen *et al.* [CPW*11] where, in addition to tracking sparse feature points like above, they define a mesh based on a Delaunay triangulation of said points. Users then define occlusion boundaries at a few key frames while the remaining frames are interpolated based on the tracked mesh.

As discussed in the following chapters, segmentation and tracking play a very important role in many video editing and synthesis applications and they are equally important in the interactive video experience scenario we strive for.

2.2.1 Video-based animation

A field of research that employs looping and jump disguise techniques but does not use video data is *data-driven character animation*. Unlike video textures, where the data is made of sequences of video frames, animation systems typically use motion capture data, *i.e.* a 3D skeleton of a tracked character with positions and orientations of joints over time. A new animation can be created, similarly to video looping, by reordering the captured data and jumping around the originally captured time-line. There has been much research in how to use this data to control the captured performance in real time. Arguably, the most important methods used to date are graph-based such as *Motion Graphs* by Kovar *et al.* [KGP02] and the similar works from the same year of Lee *et al.* [LCR*02] and Arikan and Forsyth [AF02]. These methods define a graph structure where nodes are sequences of motion data and edges represent transitions between them much like they were defined in Video Textures [SSSE00] as pointed out by Lee *et al.* in [LCR*02]. Once the graph is constructed, it is traversed based on user-defined constraints and requirements (*e.g.* follow this direction using this specific gait) to create new animations such as indefinitely looping specific ones or seamlessly transition between them.

Based on similar concepts, video-based character animation methods emerged over the years. In the original video textures paper [SSSE00], Schödl *et al.* adapted their looping technology to allow users to control the path of a filmed fish by simply using a mouse. The video of the fish is segmented, a sprite is extracted and is annotated with a velocity vector denoting where the fish is going. This information is then used to rearrange the video frames such that the fish goes towards the user-defined destination. In their follow-up paper Schödl and Essa [SE02] facilitate the animation of characters, specifically animals (*e.g.* hamsters and flies), given a video showing them naturally moving in front of a green screen. A video sprite of the filmed character is extracted from the background using chroma keying and corrections for perspective distortions are applied. Features, such as sprite velocity, area and color, are extracted and, following methods outlined in their previous work [SE01], a distance metric is learned using a linear classifier. It is then used to define transition costs between different frames in the sprite sequence which

are finally animated by optimizing a cost function using *repeated subsequence replacement*. The cost function is defined based on a set of constraints that users might want to be able to control such as location, path to follow, collisions and range of frames. Finally, the sprite sequence can be composited onto different backgrounds and projected into a virtual camera to get the desired perspective effect.

More recently, the very relevant "Human Video Textures" by Flagg *et al.* [FNZ*09] introduces video-based character animation of humans in a manner much similar to [SSSE00] while Casas *et al.* [CVCH14] achieve similar results in free-viewpoint interactive settings. They introduce a novel compression technique to cope with large amounts of input data and a new rendering technique based on optical flow for aligning texture from multiple viewpoints to avoid ghosting artefacts.

The methods above have been proven very successful given enough input data [SE02]. They excel at what they were specifically designed to do, *i.e.* perform character animation where users are interested in making the character move from a point A to a point B. It is unclear however if they would lend themselves to different types of interactions with video content. Moreover, they do not support active user or content creator involvement for fine-grained creative control or to correct mistakes.

2.3 Multi-view from Single view

The majority of the methods presented in the previous sections reason about filmed content in a two-dimensional sense. In some, time represents the third dimension, but the input videos are almost always manipulated as representations of a 2D world. In reality of course, videos are two-dimensional projections of a 3D world changing over time. By reasoning this way, many new ways of interacting with filmed content become available, such as intuitive and seamless object manipulation and new view synthesis. In this section, we present a number of works found in the literature that enable this more accurate kind of reasoning and, eventually, the more immersive and compelling interactive video experiences that we discuss in Chapter 5.

Single image-based new view synthesis

There are generally two distinct ways of generating a new view from a single image. The first one is concerned with finding geometric representations of the captured scene and using it to approximate how the scene or object would look like from a novel view. Horry *et al.* [HAA97] rely on the user to solve this problem by providing a tool that allows them to quickly define a rough approximation of the scene geometry. They define vanishing points, a set of planes for the background and billboards for the foreground objects and the resulting mesh is used to render a new, user-specified view point. Oh *et al.* [OCDD01] take a similar user-heavy approach, but in addition to geometric primitives such as planes, they allow users to also directly "paint depth" using a brush. In *Automatic Photo Pop-up* [HEH05], Hoiem *et al.* construct a 3D representation of outdoor scenes by classifying each super-pixel in an over-segmentation of the input image as one of three classes: ground plane, orthogonal walls and sky. A novel view can be rendered by texture mapping a three-dimensional mesh built on top of the classified set of super-pixels. Zheng *et al.* [ZCC*12] assume that most objects in a scene can be represented using cuboids. They can fit these simple shapes semi-automatically to objects in the scene and allow their manipulation directly in the image space. In contrast, Chen *et al.* [CZS*13] allow users to quickly model objects in an image by constraining it to fit to the image edges. This allows shapes to be defined very quickly using only a few strokes. Finally, Kholgade *et al.* [KSES14] can render new views of objects in a single image by selecting a suitably similar 3D model from a database and semi-automatically aligning it to the object itself. Similarly, Rematas *et al.* [RNR*17] present a method that can generate plausible new viewpoints of a photographed object by using the 3D information of and aligned 3D template to in-paint dis-occluded regions.

The methods above balance between detailed shape representations and necessary user effort. On the one hand, there are methods such as [ZCC*12, HEH05] which assume a simplistic representation but only require limited user input. On the other, [KSES14] require users to invest more effort but the shape representation is extremely detailed. Moreover, various assumptions are made such as the image only having a single vanishing point [HAA97] or having and finding a suitable model in a database [KSES14]. It is clear then that there is a need for

more robust and generic methods.

The second type of methods used to tackle the view synthesis problem from a single image is based on learning. The assumption is that one can learn a parametric model of a certain type of objects or scenes and use it at test time to generate a new view given an input image. An emerging technology used in learning problems are Convolutional Neural Networks (CNN). For instance, Eigen and Fergus [EF15] use a CNN to infer depth and surface normals which can be used to render a novel view of a scene. Tatarchenko *et al.* [TDB15] also use a similar approach but are able to render a novel view of a previously unseen car from just a single example image. Zhou *et al.* [ZTS*16] improve on their result by generating a flow image to displace pixels from the input image rather than directly inferring pixel colors. Not specifically designed for novel view synthesis, [MAFK17] by Mousavian *et al.* uses CNNs to estimate the bounding volume placed in the 3D world given a photo and a bounding box showing where the object of interest is in image space. This information could then be used given multiple observations of the same object to reconstruct and render it from novel view points.

The methods above have been proven to reach very impressive results. However, while [TDB15, ZTS*16] allow for high customization of captured objects in the shape of novel view synthesis, they only operate at very low resolutions. On the other hand, systems such as [MAFK17] are mostly designed for and evaluated on tracking benchmarks and, as such, may simply become a piece in the puzzle for an interactive video experience system.

Multi image and video-based new view synthesis

The papers described above suffer from being severely under-constrained. As such, multiple assumptions must be made and often one must rely on user input. One way to counteract this issue is to capture more data in order to introduce more information and potentially reduce observation noise through redundancy. Two ways to do this found in the literatures are to i) use a single but moving camera and ii) use multiple synchronized static cameras.

Casas *et al.* [CVCH14] introduce a method for creating free viewpoint animations of people performing various actions. They capture synchronized footage

from multiple static locations and use them to reconstruct the 3D shape of people. They can create new animations by interpolating between captured shapes which are textured by blending colors from multiple input viewpoints using an on-line optical flow-based rendering technique. In contrast, van den Hengel *et al.* use a moving camera to film a static scene in [vdHDT*07]. Objects present in the captured video can be modeled by manually drawing lines on the video frames using their user interface. This results in a static textured mesh that can be rendered from any viewpoint.

Different approaches are based on tracking clouds of feature points over time. Their 3D location can be triangulated as part of a process called *Structure from Motion* (SfM). For instance, Lebeda *et al.* [LHB15] reconstruct objects, such as cars, filmed with a single moving camera by tracking features and camera pose in time to produce a probabilistic model of shape. This shape is improved upon as more of the object is seen over time and results in a textured model which can be viewed from any viewpoint. In [CCM16], Chang *et al.* render novel views of filmed objects using a novel image-based rendering algorithm. They first model the scene using a set of sparse structured points which are grouped interactively by an user. Optionally, they can define primitives to better model objects. They can then define new paths for the object to follow in the captured (or a new) scene.

Unfortunately, the methods presented above suffer from a number of limitations. Methods such as [CVCH14, CCM16] require specialized capture setups and therefore are not suitable to casual scenarios and require careful planning. Additionally, [vdHDT*07, CVCH14] expect users to invest significant effort to model objects and to guide synthesis. Moreover, all methods but [CVCH14] result in static representations of an object. Thanks to their image-based rendering approach, Chang *et al.* [CCM16] can render view-dependent effects such as specularities given the right capture setup, but they cannot retain object-specific details such as the spinning wheels of a car. Finally, methods based on feature tracking such as [LHB15, CCM16] rely on the ability to match points in different views based on their appearance. While Lebeda *et al.* show in [LHB15] that this can be done for relatively small objects that occupy between 1% and 10% of the frame area, point-based features may not be as successful for smaller objects that

frequently occur in surveillance-type scenarios (which we will see make for great interactive video experiences in Chapter 4).

2.3.1 Image-based Rendering

In this section, we briefly introduce concepts and seminal papers in the field of *Image-based Rendering* (IBR). While the aim of this thesis is not to innovate in this field, IBR enables new and compelling interactive video experiences which we describe in Section 5.3.2. For this reason, we only focus on what we believe is necessary for a basic understanding and defer the reader to [FH*15] for reconstruction and the excellent image based rendering survey [SCK08].

The core idea behind IBR is the fact that photos of an object capture their appearance from a number of viewpoints. Given the shape of a scene, one can then interpolate between the input views to look at it from a novel viewing location. Roughly speaking, the difference between various IBR methods consists in the choice of reconstruction method which infers the shape of what is filmed and the choice of how to use this information to infer the appearance from a novel viewpoint.

The aim of the reconstruction stage in an IBR system is twofold: i) recover the pose of the cameras that took the photos and ii) construct a detailed representation of the scene geometry. Typically, *Structure from Motion* (SfM) is used to compute the camera parameters, *i.e.* location, orientation and intrinsics parameters. The SfM process consists of first finding 2D feature points in ever input image and matching them with one another. This results in a set of 2D tracks that show how points move between input viewpoints in the image plane. Finally, camera parameters and the triangulated 3D locations w.r.t. the cameras for all the matched points are recovered using geometric solvers and RANSAC [FB81]. An additional *Bundle Adjustment* stage may be performed to refine the initial estimates by minimizing the re-projection error of tracked points into the input image. Many SfM methods have been devised over the years, most notably [SSS08, MMMO, SF16].

Given the camera parameters and the *sparse* set of points recovered through SfM, the detailed *dense* geometry of the captured scene is constructed using

algorithms collectively known as *Multi-view Stereo* (MVS). The goal of this sections is not to discuss the in-depth intricacies of MVS, so we will limit ourselves to saying that the biggest difference between Multi-view Stereo algorithms is in the type of output. As mentioned previously, this is a dense representation of the scene geometry and it can take the form of per-pixel depth maps, clouds of 3D points with color, signed distance functions over a three-dimensional grid of voxels or triangular (textured) meshes. For more information, please consult the great practical introduction to MVS by Furukawa and Hernandez [FH*15].

Image-based Rendering (IBR) algorithms aim to render the appearance of a scene from a novel view point by only using the information captured by real photographs. The most common way to perform IBR is to use the camera parameters inferred through SfM and the dense reconstruction from MVS to infer the color at each 3D world point as a combination of their color in the input views. The amount each image contributes to the final color depends on the relationship between the input viewpoints and the new viewing location, *e.g.* the angle between the rays going from each input camera to a given point and the ray from it to the target camera. The main difference between IBR methods lay in how they balance geometry quality with the number of input images used for rendering. At one extreme of the spectrum, method such as [GGSC96] require a large number of images to blend between and can cope with very approximate geometry. At the other extreme, methods such as [HASK17] rely on very accurate geometry but only use few images to render a new viewpoint by texture mapping a mesh. Most other methods are somewhere in-between and attempt to compensate for inaccurate geometry by blending between multiple images [CDSHD13, HRDB16]. Other differences include the way input images are captured (*e.g.* narrow baseline and structured [GGSC96] or wide baseline and unstructured [BBM*01]), the type of scenes (*e.g.* outdoors [CDSHD13] or indoors [HRDB16]), the type of scene geometry (*e.g.* global [HASK17] or per-image [HRDB16]) and the supported amount of deviation from input views (*e.g.* little in-between views [GGSC96] or completely free [HRDB16]). For more details, please the comprehensive analysis of IBR techniques in [SCK08].

2.4 Conclusions

Given the vast literature presented in the previous sections, we believe there are two common limitations that need addressing.

The first limitation is the fact that the majority of the previous work has made assumptions on the type of input videos, *e.g.* static monocular cameras, and filmed objects, *e.g.* one [SSSE00, BAAR12] or few [TPSK11, JMD*12] non-overlapping ones or presenting little in-frame movement [KSE*03, LJH13]. While multi-camera or multi-sensor footage is not the focus of this thesis, by relaxing the assumptions made on the type of filmed objects we can cater for a wider variety of video types. We tackle this limitation mainly in Chapter 4 by introducing interactive tools for processing videos and using generic and abstract information to describe the filmed content.

The second, crucial limitation evident in existing literature is the lack of interactivity. This is present at two levels. On the one hand, often there is no means for people to correct automatic algorithms such as the video looping from [SSSE00, LJH13]. We tackle this in, for instance, Chapter 4 by providing user-in-the-loop procedures for tracking and segmenting objects and by defining incompatibilities between actions performed by different filmed subjects. On the other hand, consumers are generally limited to being simple spectators. For instance, once a loop is created in [SSSE00], we can merely watch the infinite video or in the case of [LJH13] use a slider to define the amount of motion. Even when more manual control is allowed by means of specialized tools [TPSK11, LZW*13], modifications are recorded and videos are rendered offline after the fact. In Chapter 4, we introduce the *action* as a generic user-defined abstraction, which we leverage to make the final video react in real time to a player's action. In Chapter 5, we introduce even more control over an input video by allowing the virtual camera to move in 3D or tracked objects to be actively controlled.

Chapter 3

Problem Analysis

Films and photos are traditionally passive mediums. People are merely spectators limited to seeing what the content creators envisioned and captured. In contrast, video games are an interactive medium where the players are an active part of how the content is enjoyed. In this chapter, we are interested in exploring ways to make traditional videos more similar to video games by instilling interactivity into them.

Over the years, many have explored ways to make photos more interactive. Typically, this is done by allowing users to move the virtual camera through a technique called image-based rendering (IBR). Techniques such as [CDSHD13, HRDB16] take multiple images of a scene, reconstruct it using Structure from Motion and Multi-view Stereo [FH*15] methods and are able to change the view point in real time by interpolating between captured ones. This allows them to virtually fly through the captured scene. Arguably, the biggest limitation of IBR methods is that they rely on scenes being static which, as a result, make them look fake even though they are technically photo-realistic.

In contrast to photos, videos capture dynamic events that happen over time which make them more compelling and immersive. However, videos are notoriously hard to work with due to the larger amount of data and the (sometimes) drastic appearance changes over time. For these reasons, traditional methods designed to work with photos of static scenes, such as the ones mentioned above, cannot typically cope with video data. Moreover, while moving the camera to reveal new parts of the scene is certainly appealing, there may be other ways to make videos

interactive and more engaging.

(a) Original video frame **(b)** In-game asset

Figure 3.1: Given a video of a candle (a), we can create a controllable video-based computer game asset (b). Note how the direction in which the flame flickers changes according to the game's wind simulation.

That said, we set out to achieve two main goals in this chapter. First, we aim to define well what interactivity means in the context of filmed content. Thanks to this PhD's setting [1], many conversations were had with game developers, the masters of designing engaging interactions. In a similar manner to how video-games are created, we asked how would we like players to interact with a video. Let us use the CANDLE video, of which we show a frame in Figure 3.1a, as a didactic example. The first, very practical issue highlighted by our conversations with game developers was the fact that, unlike video games, videos must have a finite length. It would be unfeasible to capture, store and process an infinitely long sequence of images. As a result, we would like to make the finite length video of the candle in Figure 3.1a appear seemingly infinite, through a process called *looping*. Moreover, looking at a video for a long time is not very engaging so we would like to give players some *means of interacting with it*. For instance, maybe we could let players blow on the candle and see the flame react realistically by flickering violently to the point of it even going out. As content creators, the game developers also highlighted the fact that they would like to be given the opportunity to manipulate and modify a video in similar ways to how they deal with traditional assets. It is crucial then to provide *tools* that creators can use while creating an interactive experience. Finally, modern video games are often

[1] This PhD was funded by CR-PLAY which aims to introduce IBR and VBR techniques into traditional game development pipelines (http://www.cr-play.eu/).

set in a three-dimensional world and players interact with them accordingly. We would like *videos to behave as a 3D entity* as well, despite them being simple two-dimensional representations of the real world. For instance, we would like to let players look at the candle from a different view point or maybe move it around convincingly (as shown in Figure 3.1b).

Given the wish list above, the second goal of this chapter is to introduce and investigate ways to make each item a reality. We dedicate the following sections to presenting techniques we have experimented with. Most are tailored to concrete needs but they proved instrumental in informing the decisions made in the later chapters of this thesis. For instance, in Section 3.1, we introduce the concept of looping which makes finite length videos such as the candle flame in Figure 3.1a seamlessly play back for an indefinite amount of time. In Section 3.2, we discuss how extra information, such as semantic knowledge about what video subjects are doing while filmed, can be leveraged to give players the power to influence what is happening on screen. We then discuss how to create new visuals from video frames in real time in Section 3.3 which is crucial for being reactive to user input and provide immediate feedback. We dedicate Sections 3.4 and 3.5 to showing how to get content creators involved in making videos interactive, while Section 3.6.1 introduces the third dimension into the traditionally 2D video synthesis pipeline. In each section, we also dive deep into some of the aspects that enable the technologies we suggest investigating for making our interactive video wish list come true.

Motivation

As previously mentioned, the choices made in this thesis often have very practical motivations as game developers were actively involved in many discussions. As such, clearly identifying a number of desirable qualities of a game asset based on filmed video content was very important. In this chapter, we introduce the features that we identified as critical for a successful "video-based game asset" or interactive video experience. We believe that without them the video medium would be confined to a more traditional, passive setting, as opposed to the highly interactive video game applications we strive for. The process through which we identified such properties along with the in-depth exploration of the issues

they rise were instrumental in informing and guiding the research presented in Chapters 4 and 5.

3.1 Indefinite Video Playback

One of the main limitations of traditional videos is that they have a finite length. In contrast, video games can be enjoyed for as long as one wishes as new events occur automatically or in response to user actions. As we cannot predict or do not wish to limit how long users might look at a video in a game-like scenario, it stands to reason that we must be able to present them with *filmed content for an indefinite amount of time*. Clearly, we cannot film an infinitely long video and long frame sequences are impractical as they cannot be trivially stored or processed.

Since the early days of the video games industry, one way to deal with memory constraints when presenting users with animations was to *loop* short clips, *i.e.* play them back from the beginning once their end is reached. For instance, a character's walking animation might be only a few frames long but, through clever planning, the last frame seamlessly transitions to the first one, giving the impression of an infinitely long sequence of frames. Traditionally, such animations are created manually by expert artists however, in 2000, Schödl *et al.* [SSSE00] introduced an automatic technique called *Video Textures*. It relies on the assumption that, given a corpus of video frames, similarly looking pairs can be found and used interchangeably. If that is the case, one can automatically and seamlessly jump to different places in the video's original time-line, giving thus the impression that it is infinitely long and varied events happen randomly.

Arguably, the most crucial part of a successful video texture is finding pairs of frames that are sufficiently similar to result in seamless transitions. To do so, we need to define a *measure of similarity* which returns a large number if two frames look alike and a small number otherwise. Unfortunately, finding such a measure is a hard problem. First, a distance measure may behave unpredictably such as demonstrated using our RIBBON dataset in Figure 3.2, where only one of the shown measures agrees with the arguably true statement "frame A is more similar to frame B than it is to frame C". Second, as we discuss in Section 3.1.2,

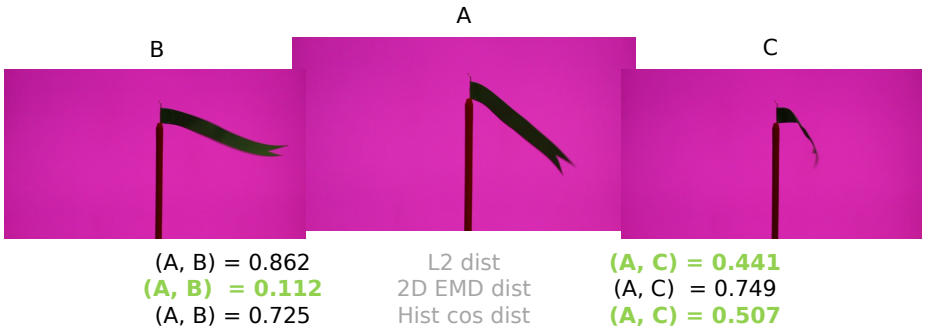


Figure 3.2: Similarity according to different distance metrics (lower is better). It is clear that A and B should be more similar than A and C but only the 2D EMD metric agrees.

there may be pairs of dissimilar frames that result in seamless transitions as witnessed by the RIBBON dataset video[1] where movement is fast and erratic. Not being able to find such pairs may prevent us from successfully looping a video. Lastly, videos may not have any similar pairs of frames such as when they capture complex moving subjects or multiple independent elements. In these cases, finding the best matching frames and morphing between them [SLWSS15] or separating them into independent groups of pixels [LJH13] may be the only way to create a successful video texture.

As it is one of the most important aspects of video looping, we now investigate a variety of similarity measures that we believe may be suitable for finding seamless transition points in arbitrary sequences. In Section 3.1.1 we focus on *objective distances*. They come mainly from mathematics and are usually defined in terms of a topological *space* such as the three-dimensional *RGB*-color space. Their strength lies in the fact that they are simple to compute and generic. One such measure is the Euclidean or $\mathbf{L}_2$ distance which has been successfully used by Schödl *et al.* [SSSE00] and many others. However, as mentioned above and shown in Figure 3.2 they may behave unpredictably and not agree with human perception. In Section 3.1.2 we investigate *perceptual distances* which try to model how humans perceive images and therefore match their expectation more closely. Arguably, the best known measure is the *Structural Similarity Index* (SSIM) developed by Wang *et al.* [WBSS04]. We focus on a special family of similarity measures

[1] https://corneliu.co.uk/phdresults/chapter3/index.html#ribbon

that rely on supervised learning methods such as Random Forests [Bre01] and user-defined examples.

3.1.1 Objective Distances

Distance metrics such as the Euclidean Distance are commonly used in the literature (*e.g.* [SSSE00, KSE*03, TPSK11]) thanks to their relative simplicity and ability to usually *just work* out of the box without complicated parameter tweaking or additional user input. Many metrics have been explored and used for a variety of purposes over the years, suggesting that they are more or less suitable depending on the application and that choosing one is an ongoing question. After a short note on memory constraints, we present below the distance metrics we have explored and evaluated.

Memory constraints Videos can be comprised of thousands of frames, so it stands to reason that we must always carefully consider memory requirements. In order to seamlessly jump between sections of a video, we must find similar pairs of frames which involves computing an $N \times N$ distance matrix D. Our ribbon dataset for instance, is made of $N = 1280$ frames each containing $1280 \times 720 \times 3 = 2764800$ pixel values. Each frame, stored as a 64-bit double array, requires approximately 22MB so we would need about $1280 \times 22\text{MB} \approx 28\text{GB}$ to keep the full dataset in memory. Conversely, we could read each frame from disk when needed (on average $N/2 = 640$ times). However, this means reading $640 \times 1280 \times 22\text{MB} \approx 18\text{TB}$ of data from disk.

To avoid such a large amount of slow disk accesses and reduce memory requirements altogether, we divide our data into blocks of size $S = N/K$. To process the ribbon dataset for instance, we need only read 7040 images from disk when $K = 8$, *i.e.* an over $100\times$ reduction in disk accesses from $\approx 18\text{TB}$ to $\approx 155\text{GB}$. Algorithm 1 is then used to compute the distance matrix.

Euclidean Distance

The Euclidean distance is commonly used in the literature because of its simplicity and represents the distance between two points in an Euclidean N-space. Given

Data: Block size S, number of images in dataset N,
images $\mathcal{I} = \{\mathtt{I}_i, i \in [0, N)\}$
forall the *images* $\mathtt{P} \in \{\mathtt{I}_i, \mathtt{I}_{i+1}, ..., \mathtt{I}_{S(i+1)}, i \in [0, K)\}$ **do**
  Compute distance matrix $d(\mathtt{P}, \mathtt{P})$;
  forall the *images* $\mathtt{Q} \in \{\mathtt{I}_j, \mathtt{I}_{j+1}, ..., \mathtt{I}_{S(j+1)}, j \in [i, K)\}$ **do**
    Compute distance matrix $d(\mathtt{P}, \mathtt{Q})$;
  end
end

Algorithm 1: Computing the distance matrix by filling the memory with one block of frames at a time.

images $\mathtt{P} = \{p_i | i \in [1, N]\}$ and $\mathtt{Q} = \{q_i | i \in [1, N]\}$, represented as concatenated pixel intensities p_i and q_i respectively, we define the Euclidean distance

$$d_E(\mathtt{P}, \mathtt{Q}) = \sqrt{\sum_{i=1}^{N} (q_i - p_i)^2} \tag{3.1}$$

between them, where $N = \text{Width} \times \text{Height} \times \text{Channels}$.

Cosine Distance

The cosine distance d_C is defined as the cosine of the angle θ between two vectors. If they are the concatenated pixel intensities of images $\mathtt{P}$ and $\mathtt{Q}$ as defined above, we compute

$$d_C(\mathtt{P}, \mathtt{Q}) = \cos(\theta) = \frac{\mathtt{P} \cdot \mathtt{Q}}{\|\mathtt{P}\| \|\mathtt{Q}\|}. \tag{3.2}$$

While this approach can suffice in certain cases, it is hard to predict the performance of the cosine distance for very high dimensional vectors such as our images. Moreover, as witnessed by the large areas of pink background present in the RIBBON dataset (Fig. 3.2), often only parts of an image are informative. Based on these observations, we explore a different representation for the contents of an image.

Assuming a foreground-background segmentation mask is provided (such as described in Section 3.4), we first divide each segmented image $\mathtt{I}$ into a grid of size $N \times M$ (shown in Fig. 3.3a). For each grid section s_i, we count the number of foreground pixels and normalize by the total number of foreground pixels so

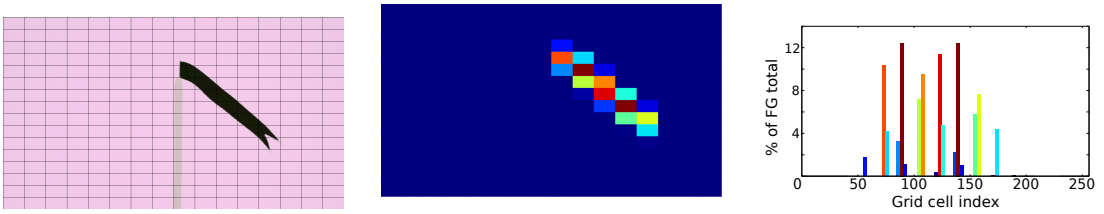
that they sum up to 1. This gives us an $N \times M$-dimensional vector

$$\mathtt{I}_G = \left[c_i, i \in [1, N \times M]\right], \quad c_i = \frac{\text{count}(s_i)}{\sum_{i=1}^{N \times M} \text{count}(s_i)}, \tag{3.3}$$

where count$(\cdot)$ indicates the aforementioned number of foreground pixels, which gives us a histogram of foreground pixels binned by grid section (Fig. 3.3c). We then define the cosine distance between images $\mathtt{P}$ and $\mathtt{Q}$ in terms of their new representations, $\mathtt{P}_G$ and $\mathtt{Q}_G$ respectively, as

$$d_C(\mathtt{P}_G, \mathtt{Q}_G) = \frac{\mathtt{P}_G \cdot \mathtt{Q}_G}{\|\mathtt{P}_G\| \|\mathtt{Q}_G\|}. \tag{3.4}$$

We have experimented with 4×4, 16×16 and 32×48 grid sizes and found the



(a) Grid division of input image with foreground mask overlayed

(b) Number of foreground pixels per grid section (**red** means higher)

(c) 1D histogram representation of the pixel counts

Figure 3.3: Visualization of the grid representation $\mathtt{I}_G$ from Equation 3.3.

first to be too low resolution to accurately describe the ribbon movement and the last to give little improvement over a 16×16 grid size (shown in Figure 3.3) while increasing computation time and memory requirements substantially.

It is worth noting that this approach is only suitable for cases where an object's movement can be characterized by its segmentation silhouette. Objects such as the pendulum in [SSSE00] that do not move as a whole w.r.t. the background would not benefit from this representation as only parts of the foreground change appearance over time. A pixel-wise color comparison would be more suitable for such cases.

2D Earth Mover's Distance

The Earth Mover's Distance (EMD) was first proposed by Rubner *et al.* [RTG98] as a similarity measure between two multi-dimensional probability distributions.

The name derives from the fact that the first distribution can be seen as "piles" of earth and the second one as "holes" to be filled. We assume that the number of "piles" and "holes" is the same and that the amount of earth and the capacity of each "hole" are proportional to the corresponding distributions. The goal is to minimize the effort needed to move the earth from "piles" i to fill the "holes" j which is determined based on i) the amount of earth that needs moving, *i.e.* the *flow* $\mathbf{F} = [f_{ij}]$, and ii) the so called *ground distance* between them $\mathbf{D} = [d_{ij}]$.

In our case, we define the EMD on the 2D grid representation defined in Equation 3.3. We follow the solution detailed by Rubner *et al.* [RTG98] by first solving a *transportation problem* [Hit41] to find the *flow* $\mathbf{F}$ that minimizes both the amount of displaced earth and the traveled distance. We then compute the 2D EMD distance

$$d_M(\mathtt{P}_G, \mathtt{Q}_G) = \frac{\sum_{i=1}^{N\times M}\sum_{i=1}^{N\times M} f_{ij}d_{ij}}{\sum_{i=1}^{N\times M}\sum_{i=1}^{N\times M} f_{ij}}, \tag{3.5}$$

where the *ground distance* $\mathbf{D}$ is the $\mathbf{L}_2$ distance and an example of $\mathtt{P}_G$ and $\mathtt{Q}_G$ can be seen in Figure 3.3b.

3.1.2 Perceptual Distances

We have previously mentioned and illustrated in Figure 3.2 that different metrics can work well in some cases and less well in others. Moreover, we are interested in the specific case of finding pairs of frames (i, j) that, when used interchangeably, result in seamless transitions between different parts of a video. While it is generally true that when frames i and j are similar, showing frames i and $j+1$ in sequence is seamless, we believe it is not a pre-requisite. This is especially true of videos depicting subjects with complex moving patterns such as our RIBBON dataset. It is then clear that our metric should be robust and flexible enough to correctly model both situations.

It is unclear how to measure the "seamlessness" of a jump accurately. Instead, we turn to the user, and while it would be ideal to ask them which exact pairs of frames result in invisible jumps when used interchangeably, the task would be prohibitive given the number of possible pair combinations. In this section, we

explore two different approaches to take only a few user-given examples and learn a distance measure that can predict what the user would think of a pair of frames they have not seen.

Random Forest Regressor

Inspired by Xiong *et al.* [XJXC12], we use a Random Forest Regressor (RFR) [Bre01] to model the user perceived similarity and predict the distance $d_F(\mathtt{P}, \mathtt{Q})$ between each pair of frames $\mathtt{P}$ and $\mathtt{Q}$. We define the mapping function

$$\phi(\mathbf{x}_\mathtt{P}, \mathbf{x}_\mathtt{Q}) = ||\mathbf{x}_\mathtt{P} - \mathbf{x}_\mathtt{Q}||_2, \tag{3.6}$$

which gives us a *frame-pair* feature representation. Here, $\mathbf{x}_\mathtt{P}$ and $\mathbf{x}_\mathtt{Q}$ are Histogram of Oriented Gradients (HOG) descriptors of their corresponding frames. First described by Dalal and Triggs [DT05], HOG features describe an image using histograms of gradient directions computed at pixels within small connected regions called cells. We set the cell size to 16×16 pixels and bin the gradient directions into $b = 8$ discrete bins (see Figure 3.4 for an example matted image from our RIBBON dataset and corresponding HOG features).

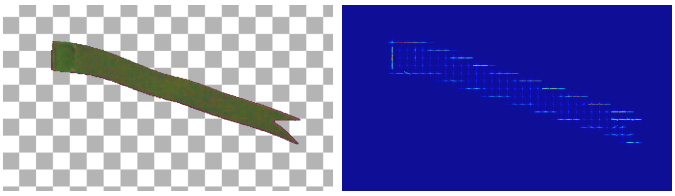


Figure 3.4: Left: Zoomed-in matted frame for our RIBBON dataset. Right: HOG features for the given frame. We use 16×16 cells and bin gradients into 8 orientations. Warmer colors, signify higher numbers of a certain orientation.

We then use Fisher Vector Encoding (FVE) [PD07] on the resulting mapping $\phi(\mathbf{x}_\mathtt{P}, \mathbf{x}_\mathtt{Q})$ which clusters the feature vectors using a Gaussian Mixture Model (GMM) trained on a set of labeled frame-pair features that defines a dictionary of *visual words*. The new frame-pair feature representation $\phi^F_{\mathtt{PQ}}$ resulting from the soft association to each *word* (*i.e.* Gaussian in the GMM) is a $2b \times c = 16 \times 10$-dimensional vector where $c = 10$ is the number of Gaussian components.

In parallel, users label random pairs of frames $(\mathtt{I}_i, \mathtt{I}_j)$ as either compatible

or incompatible. Instead of a more traditional side-by-side comparison, users inspect sequences of frames $\{\mathtt{I}_{i-k}, ..., \mathtt{I}_{i-1}, \mathtt{I}_i, \mathtt{I}_{j+1}, \mathtt{I}_{j+2}, ..., \mathtt{I}_{j+k}\}$ to more easily decide whether a transition is noticeable. Active Learning approaches such as [MACKB14] could be used to predict which pairs are most informative at any time and to minimize the number of pairs that need to be labeled manually (approx. 150 in our experiments). We then train the RFR using these examples and the corresponding $\phi^F_{\mathtt{PQ}}$ frame-pair feature representation to regress $d_F(\mathtt{P}, \mathtt{Q})$.

Weighted $\mathbf{L}_2$ Distance

Up until now, we have assumed every region of an image, whether a single pixel or a patch, is equally informative to a measure of similarity. Typically, there are many dynamic elements in a video frame, such as the fluttering flag, the people passing by or the clouds moving in the sky seen in Figure 3.15. Clearly, many distinct elements, or *features*, can influence the measure of similarity between two frames. For the metrics described above, we have manually chosen the features we were interested in (*e.g.* how much of a certain region is part of the foreground) with varying success. What if we could choose what to "care about" automatically?

Given the above intuition, we propose a metric that roughly falls under the category of feature selection methods but with a twist. If we represent each video frame by a number of representative features, we postulate that the $\mathbf{L}_2$ distance in Equation 3.1 is expressive enough for our purposes but only a subset of these features are relevant. Intuitively, if a frame is represented by concatenated pixel intensity values, we would like to find which are more informative for judging similarity.

We approach this problem from a linear regression standpoint and aim to estimate a world state w given observed measurements $\mathbf{y}$. In our case, w represents the measure of similarity and $\mathbf{y}$ is the feature representation of a pair of frames. As before, we define a mapping function

$$f(\mathbf{x}_\mathtt{P}, \mathbf{x}_\mathtt{Q}) = \left[(\mathbf{x}_\mathtt{P} - \mathbf{x}_\mathtt{Q})^2\right] = \mathbf{y}, \tag{3.7}$$

where again $\mathbf{x}_\mathtt{P}$ and $\mathbf{x}_\mathtt{Q}$ are feature descriptors of frames $\mathtt{P}$ and $\mathtt{Q}$ such as concatenated pixel intensity values. We then define the linear regression-based distance

as

$$d_W(\mathtt{P}, \mathtt{Q}) = \sqrt{\boldsymbol{\phi}\mathbf{y}}, \tag{3.8}$$

where $\boldsymbol{\phi}$ weighs the importance of each feature y. Crucially, when there are little training examples and we are unsure of how similar frames should look like, $\boldsymbol{\phi}$ is mainly composed of 1s and $d_W(\mathtt{P}, \mathtt{Q})$ reduces to the original $\mathbf{L}_2$ distance.

To find $\boldsymbol{\phi}$, we use Linear Regression to maximize

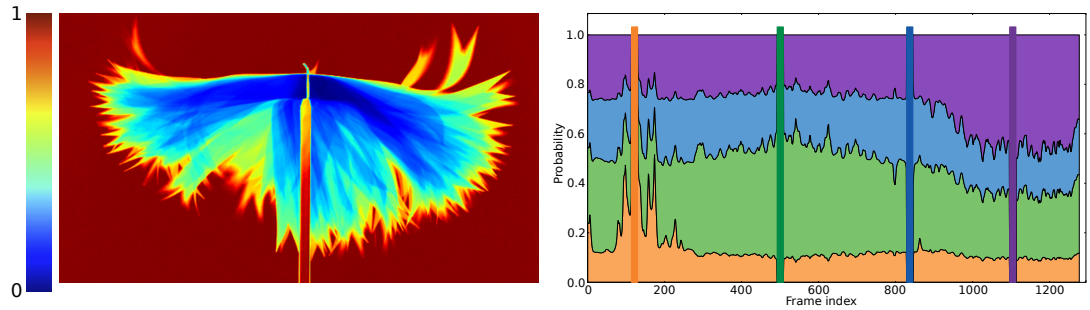
$$\boldsymbol{\phi} = \arg\max_{\boldsymbol{\phi}} \left[\prod_{i=1}^{I} Pr(w_i|\mathbf{y}_i, \boldsymbol{\phi}) Pr(\boldsymbol{\phi}|\alpha, \beta) \right], \tag{3.9}$$

where the likelihood $Pr(w_i|\mathbf{y}_i, \boldsymbol{\phi}) = \mathcal{N}_{w_i}(\mathbf{y}_i^T\boldsymbol{\phi}, \sigma^2)$ is modeled using a normal distribution while the prior $Pr(\boldsymbol{\phi}|\alpha, \beta) = \mathcal{G}_{\boldsymbol{\phi}}(\alpha, \beta)$ is a gamma distribution with $\alpha = 5, \beta = 6$. Intuitively, the prior favors weights $\boldsymbol{\phi} = \mathbf{1}$ as we want $d_W(\mathtt{P}, \mathtt{Q})$ to return the $\mathbf{L}_2$ distance when few examples are available. Writing down the full normal and gamma distributions definition, the function to maximize becomes

$$\boldsymbol{\phi} = \arg\max_{\boldsymbol{\phi}} \left[\frac{1}{2\sigma^2} \sum_{i=1}^{I} (w_i - \mathbf{y}_i^T\boldsymbol{\phi})^2 + \sum_{d=1}^{D} \left[(\alpha - 1)\log(\phi_d) - \beta\phi_d \right] \right], \tag{3.10}$$

where we took the logarithm of the probabilities in 3.9 to turn the products into sums. We then use an off-the-shelf gradient descent implementation to solve for $\boldsymbol{\phi}$. Figure 3.5a shows the resulting values of $\boldsymbol{\phi}$ regressed using the same 150 examples we used to train the Random Forests Regressor in the previous section. It is worth noting how the pixels showing the tip of the ribbon are assigned a higher weight, while the pixels showing the background, which is not changing in the original footage are assigned a value of 1 as their $\mathbf{L}_2$ distance is close to 0 so changing their associated weights does not influence the maximum value of $\boldsymbol{\phi}$ in Eq. 3.9.

As previously mentioned, for this experiment, we have used the concatenated pixel intensity values as a feature descriptor $\mathbf{x}$ for each input image. It would be interesting to test more diverse features such as HOG, optical flow and others. We have not however tested this further as the Euclidean distance proved successful for our use cases as we will see in Chapter 4.



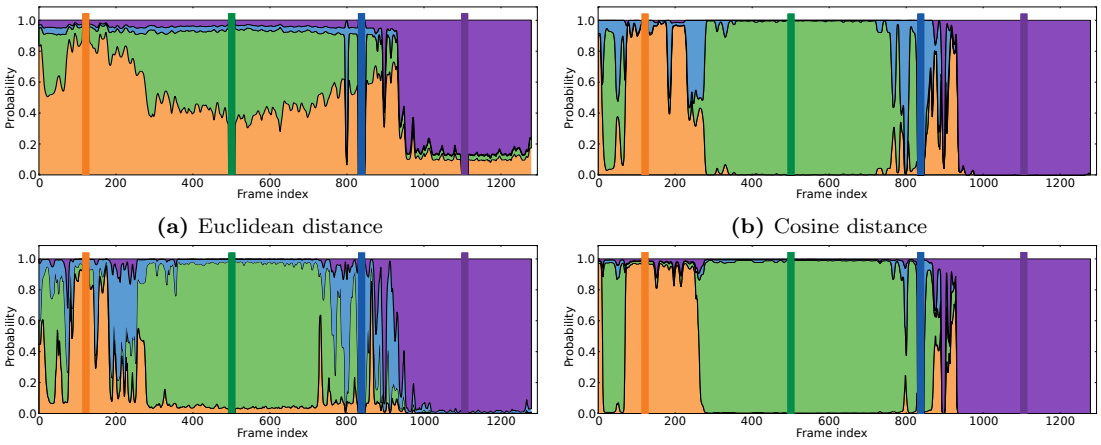
(a) Reshaped values of ϕ **(b)** Propagated label probabilities

Figure 3.5: In (a), we show the values of $\phi \in [0, 1]$ from Equation 3.10 regressed for the RIBBON dataset. We use the same user-given examples used for training the RFR metric to regress ϕ which we have reshaped to the input image size for visualization purposes. The propagated action label probabilities in (b) can be compared to the ground truth labels in Figure 3.7 for qualitatively assessing accuracy.

3.1.3 Considerations

In this section, we have presented a number of similarity metrics. In order to gauge their effectiveness at finding similar pairs of frames, we have tested them on the task of label propagation as described in Section 4.2.3. The reason for this choice is twofold: i) defining ground truth association between labels and video frames ;-is much more reliable than defining seamless jumps and ii) the two problems are essentially equivalent as they are designed to find visually similar pairs of frames. We perform all the tests on our RIBBON dataset as it is simple to define ground truth labels (see Figure 3.7) while still being challenging as witnessed by the example in Figure 3.2 and the visual ambiguity of the **orange** and **cyan** classes from Figure 3.7. We run label propagation [ZGL*03] using each similarity metric and the same 17 examples per class as input and show qualitative comparisons in Figures 3.6 and 3.5.

The Random Forest Regressor measure d_F, shown in Figure 3.6d, proves best when compared to the manually defined ground truth shown in Figure 3.7. It shows much less noise than the alternatives and as discussed previously it is the most general as it is agnostic to the type of data they are given. On the other hand, the weighted $\mathbf{L}_2$ measure is by far the least successful when comparing the results in Figure 3.5 to the provided ground truth (Fig. 3.7). We believe this is



(a) Euclidean distance **(b)** Cosine distance

(c) 2D EMD **(d)** Random Forests

Figure 3.6: Label propagation results given the same labeled training examples (colored vertical lines) and different distance metrics. Qualitatively comparing these results to the ground truth labels in Figure 3.7 suggests the Random Forests metric in (d) is the most accurate.

due to our choice of feature descriptor (*i.e.* concatenated pixel intensities) which is very high dimensional and perhaps not informative enough for our task. Future investigation into different descriptors such as the HOG features we used for the RFR-based d_F measure may lead to more accurate labeling. Finally, the EMD distance d_M and the cosine distance d_C show promise, however they are rather specific to the RIBBON dataset due to the grid features we define in Section 3.1.1, so may not adapt well to different videos. Despite the promising results shown by d_F, like many others before us, we choose to use the simpler Euclidean distance for our system described in Chapter 4 as it does not require training examples or any other user input. Instead, we choose to design a responsive user interface to make it as easy and fast as possible for users to define as many examples as it takes to overcome the limitations of the $\mathbf{L}_2$ distance.

3.2 Controlling Video Output

The introduction of video textures by Schödl *et al.* [SSSE00] was a very important step towards accomplishing the goals of this thesis. However, one of the main reasons that has kept content creators from actively using this technology is the

lack of control over the synthesis process (*e.g.* to correct mistakes or influence the result). Moreover, users are still passively enjoying a video, albeit randomly varying and seemingly infinite, without any way of interacting with it. It is clear then that we need some way of *influencing or guiding a video synthesis process* such as video textures. This would allow content creators to build custom experiences such as, for instance, synchronizing a video to music and, if the process is real time (such as described in Section 3.3), users to feel more engaged as they receive immediate visual feedback based on their active input.

In the original video textures paper [SSSE00], without any additional information, the output video can only be completely randomized (as long as seamless transitions are guaranteed). It turns out that, in order to enable more control over the output, it is crucial to add extra information into the mix. For instance, Schödl *et al.* add the knowledge that variations of a dynamic event are filmed sequentially so the output can be forced to only contain a subset of the input frames. This allows them to, for instance, change the speed at which a person is running on a treadmill. Many other attempts have been made over the years with interesting applications to character animation [SE02, FNZ*09] or video editing [BSHK04, BAAR12, JMD*12] and they all rely on introducing additional information other than time progression. We now discuss two distinct types of information we can introduce into video synthesis for more meaningful interactive video experiences.

3.2.1 Semantic Looping

As mentioned above, there are ways to add control to video synthesis but we argue they are either too specialized, such as [FNZ*09], or too abstract as seen in [JMD*12]. We believe we need an abstraction from the video frames that is flexible and robust but at the same time intuitive and easy to define. This abstraction must be *flexible* to cater for a variety of videos and *simple* to enable new and creative ways of guiding synthesis. For instance, we might want to control synthesis using a traditional keyboard and mouse pair. Or we might want to change the synthesized sequence based on events triggered by gameplay such as the candle in Figure 3.1b. We might even want to synthesize a new video based

on another video by using the texture-by-numbers paradigm as in [HJO*01] or the innovative makeymakey [Joy12].

In order to reach our goal, we introduce the concept of semantics which we use to constrain the looping algorithm. As previously mentioned, looping amounts to reshuffling the input video frames, but, in addition to ensuring seamless transitions, we want to enable transitions to semantically meaningful sections of the input video. This concept can be better understood with an example. Figure 3.7 shows four frames of a video showing a RIBBON fluttering as the "wind" blows. For

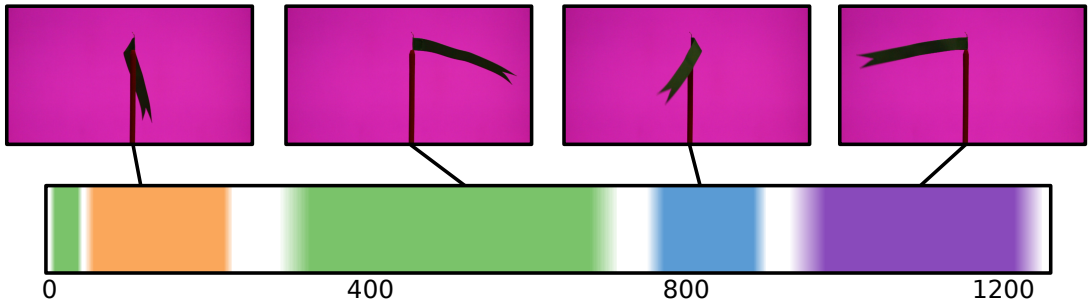


Figure 3.7: Sample frames of the RIBBON video dataset with annotated time-line. Each color represents one of four different classes of motion: ribbon fluttering **away**, **right of**, **towards** and **left of** the camera.

a more interesting outcome, we move the fan used to create the air flow in a circular manner around the ribbon. This results in video frames showing the ribbon fluttering in four distinct directions: away from the camera, from left to right, towards the camera and from right to left. Please see the full input video on our supplemental website[1]. At the bottom of Figure 3.7, we marked where in the time-line the visualized frames come from and color coded the four classes of motion as **orange**, **green**, **cyan** and **purple** respectively. If we were to loop the sequence using [SSSE00], we could randomly reshuffle the frames or, since we moved the fan around the ribbon in a sequential manner, we could manually select between the handful of subsets of frames showing the different motions. But what if we wanted a more realistic scenario and filmed the ribbon in the wild where we would not have direct control over the unpredictable wind direction? The time-line in Figure 3.7 would suddenly not look as neat and there would be

[1] https://corneliu.co.uk/phdresults/chapter3/index.html#ribbon

colors everywhere, making the method in [SSSE00] unsuitable.

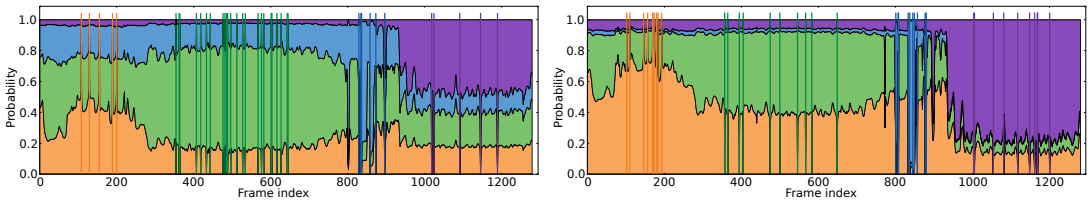
For a more intuitive interaction, we note that we associate frames to one another not based on how close in time they have been captured (as done by [SSSE00]) but based on whether the ribbon is performing the same *semantic action*, in this example fluttering in the same direction. Here, we do not mean semantic in the more commonly accepted meaning of the word, but rather we refer to the fact that the visual appearance of the ribbon is comparable in frames where it is seen performing the same action. If we can associate an abstract meaning to every frame, we are suddenly free from the constraints of filming events sequentially and have a very powerful and intuitive way of controlling what we want to show in the output video.

We believe that associating abstract *semantic* meaning to a filmed sequence is the key to having better control over the synthesis process. It can be easily understood and leveraged by content creators for creative video editing and synthesis. It also gives us the flexibility we strive for as it can be applied to multiple scenarios despite its simplicity. In fact, not only flags and ribbons lend themselves to semantic actions. We could apply the same reasoning to the intensity of a flame, people sitting or standing, cars crossing intersections and stopping at traffic lights. There are many other dynamic events that can associated to a finite number of semantically meaningful and visually distinct actions, but how can we practically leverage this abstraction?

Our goal is to associate a label corresponding to an action to each video frame. We would also like to do this in an interactive way as we can ensure better and more meaningful results by involving the user in the process. On the one hand, if we were to assume actions were filmed sequentially, we could easily manually mark subsets of frames as done in motion capture systems such as [KGP02, LCR*02, AF02], but this would preclude us from using casually captured videos and require careful planning. On the other hand, we would like to avoid having users manually label each frame as it would quickly become impractical for videos containing hundreds or thousands of stills. To reduce the required amount of manual labor while maintaining flexibility and catering for a large variety of videos we adopt a semi-supervised learning approach. The basic idea is to have users manually label only a few frames per action class and label the remaining frames automatically

based on visual similarity. We further discuss the technical side of this process called label propagation [ZGL*03] in Section 4.2.3, but for now it is important to realize that it groups the set of input frames into clusters of visually similar images. We can leverage this information to constrain the video synthesis process and introduce the type of control we strive for. For a full frame-to-action association example for a similar dataset to the RIBBON discussed in this section, please see the CANDLE dataset from Figure 3.1 on our supplemental website[1].

Choosing the right frames One critical aspect in involving the user in the process of action definition, is the choice of frames we ask them to label. If they are chosen randomly, classes might be sampled in different ways, so there would be more examples for some of them and less for others, requiring users to label a large number of frames for acceptable results. Figure 3.8 shows that, given the same number of labeled examples, we can get much better class separation if we carefully choose our examples (Fig. 3.8b) as opposed to presenting users with randomly selected frames (3.8a). To prove this point, we *cheated* by choosing



(a) Randomly chosen training examples **(b)** Training examples chosen from known intervals

Figure 3.8: The effect of better training examples. In (a), training examples (vertical colored lines) are chosen randomly resulting in 5, 24, 6 and 5 examples for the four classes (**orange**, **green**, **cyan** and **purple**) respectively. In (b), examples are chosen from known intervals resulting in 10 examples for each class. It is clear that by training on better examples, the **purple** class can be better distinguished from the rest and the **orange** class is better distinguished from the **green** and **cyan** ones. Note that these results were produced using the Euclidean distance in Equation 3.1 and ground truth labels for a qualitative assessment of accuracy are shown in Figure 3.7

the same amount of frames for each class as we already know where they are in the input video (see Fig 3.7). Active learning methods such as [MACKB14] can

[1] https://corneliu.co.uk/phdresults/chapter3/index.html#candle

automatically select the most informative examples to label, However, we show in Chapter 4 that a responsive user interfaces is a viable alternative where users select the frames interactively based on immediate visual feedback from speedy label propagation methods such as [ZGL*03].

Choosing the right similarity measure Another aspect that we need to consider is which measure of similarity to use. The Euclidean distance used by video textures [SSSE00] seems to sometimes struggle distinguishing even between clearly different classes. Moreover, such a metric might not be appropriate for class labeling making finding an ad hoc one necessary. We dedicate Section 3.1 to finding alternative distance measures.

3.2.2 Speed Normalization

The visually distinct semantics-based manipulation described in the previous section is very well suited for representing a discrete set of actions (and to some extent continuous transitions between them as we show in Section 4.2.3). However, it is not as well suited to describing continuous properties of dynamic elements such as moving direction or speed. For instance, given a video of a road crossing where multiple cars are passing by, could we give users control over a car's speed, when to stop at a traffic light or when and by how much to turn?

In this section, we investigate a different way of giving users control over the synthesis process of a new video. We allow them to control the movement speed of objects filmed in a scene. This enables the creation of our video-based computer game prototype called Counter Loop described in detail in Section 4.6.1. We assume that the input video is filmed with a static camera, that the objects of interest are moving on a planar surface and that they are associated to a two-dimensional bounding box tracked over time. The simplest way to give players control over the moving cars filmed in our HAVANA dataset (see Fig. 3.10) and used to create Counter Loop would be to segment them from the background using the tracked boxes as a cue and then let the users manipulate the speed at which we play back the frames. However, the resulting animations would likely show the cars unpredictably slowing down or speeding up depending on their

movement in the filmed footage. Since we had no control over the cars during filming, the relationship between their speed and the passing of time (*i.e.* what players can effectively manipulate) is not constant.

To tackle the above problem, we must remove the time independent effects and synthesize sequences of frames for each car where their speed appears to be constant, the same way time passes. We call this process *speed normalization* as it effectively normalizes the speed of every moving object to the same arbitrary constant in a similar manner to how we capture an image every 33 milliseconds when filming a 30 frames-per-second video. To better understand the problem at hand, take the example in Figure 3.9. The trajectory $\mathbf{T} = \left[\mathbf{x}_t, \forall t\right]$ is a set of

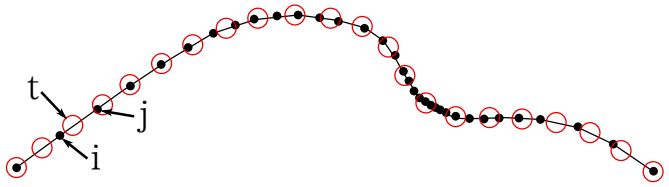


Figure 3.9: Trajectory consisting of a set of 2D points shown as black dots. It is regularly sampled into equally-sized segments by the red circles. Please see the text where we discuss the highlighted point in the re-sampled trajectory t and its two closest points in the original trajectory i and j.

two-dimensional points represented as column vectors $\mathbf{x}_t = \begin{bmatrix} x \\ y \end{bmatrix}$ (black dots in Fig. 3.9) defined at each video frame t. It is clear to see that, the slower the object moves the more packed the trajectory points are. One option for defining the points $\mathbf{x}_t$ is to take the center of the tracked 2D box we have for an object at time t. However, due to the perspective transformation that occurs when capturing the 3D world through a video camera, the amount of movement of an object in image space does not necessarily correspond to their speed in the real world. In fact, the amount of space an object appears to move in an image is proportional not only to their speed but also to their distance to the camera. Slow objects that are close to the camera could appear to be moving faster than fast objects that are very far away. It stands to reason then that we must model the perspective transformation that filmed objects are subjected to in order to be able to correctly estimate their speed and realistically depict their movement when players control them.

The simplest way to model a perspective transformation between two planar surfaces (the ground and image planes) is by means of a *homography* $\mathbf{H}$. To estimate the 3×3 matrix $\mathbf{H}$ we follow the closed form solution described in [Pri12] which can robustly map a rectangle to any other rectangle. The transformed trajectory $\mathbf{T}' = \mathbf{HT}$ then consists of 2D points $\mathbf{x}'$ (the z-axis coordinate is set to 0) on the real-world three-dimensional ground plane the object moves on. This removes the perspective projection effects described above and gives us a more accurate depiction of an object's movement.

Given the trajectory $\mathbf{T}'$, we regularly sample it to yield the constant speed trajectory $\hat{\mathbf{T}}'$ shown in Figure 3.9 as red circles. We are now almost ready to synthesize a car traveling at a constant speed. At each time step, we could compute the position of the car on the 3D-world ground plane (*e.g.* red circle denoted by t in Fig. 3.9) and show the original frame where the car is found closest to the desired location (black dot denoted by i in Fig. 3.9). In cases where there are no such frames, especially when the car moves faster than the desired speed as visible at the extremes of the trajectory in Figure 3.9, the result will look unnatural. To prevent this from happening, we estimate the appearance of the car as if it was filmed at time t by interpolating between the two closest original frames (black dots denoted by i and j in Fig. 3.9). Similar to [Wol90], we first compute the forward and backwards optical flow at each pixel location s, $\mathcal{F}_{i \to j}(s)$ and $\mathcal{F}_{i \leftarrow j}(s)$ respectively, between frames i and j. We then compute the appearance of the object at location t by defining the color $\mathbf{X}(s)$ at pixel location s as

$$\mathbf{X}(s) = \frac{d_{it}}{d_{ij}} \mathcal{F}_{i \to j}(s) + \frac{d_{tj}}{d_{ij}} \mathcal{F}_{i \leftarrow j}(s), \tag{3.11}$$

where d_{ij} is the distance between locations i and j and d_{it} and d_{tj} are defined in a similar manner. An example input sequence[1] and resulting constant speed video[2] can be seen on our website.

Note on implementation: In order to create the Counter Loop video game described in Section 4.6.1 we involved game developers in the processing

[1] https://corneliu.co.uk/phdresults/chapter3/index.html#inputspeednorm
[2] https://corneliu.co.uk/phdresults/chapter3/index.html#outputspeednorm

of the input video and creation of the necessary video assets (*i.e.* segmented car sprites tracked over time and with their speed normalized). To achieve this, we implemented a purpose-built tool (see Figure 3.10) which integrates all the algorithms described above. The content creators are able to quickly define the

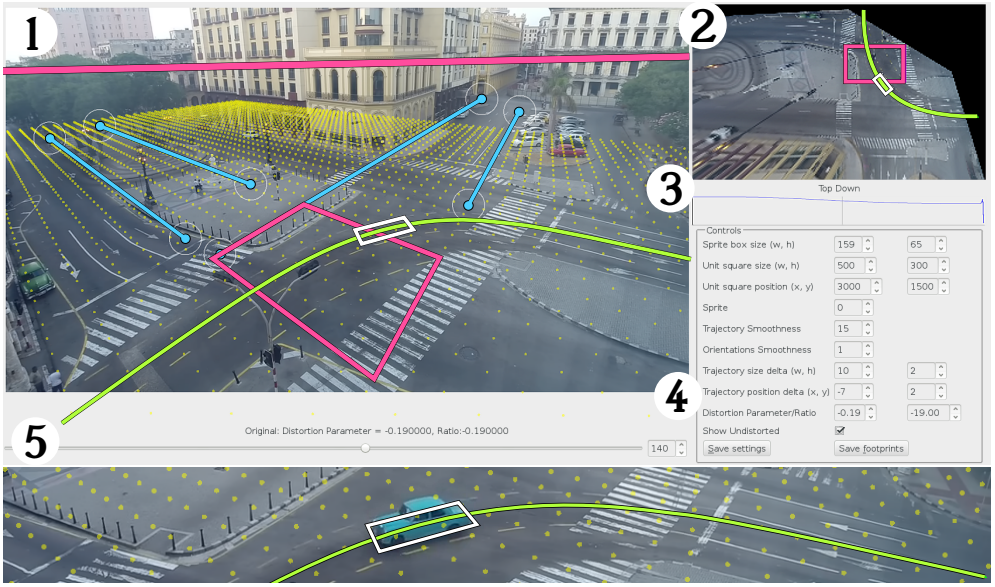
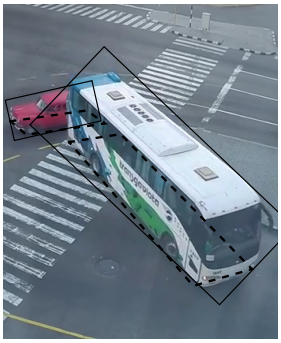


Figure 3.10: Our camera parameter estimation and speed normalization tool. (**1**) *main viewer*: users place the two pairs of **blue** lines by dragging the circular handles at their extremes; the camera parameters are defined by estimating the homography **H** between the **red** image-space intersection rectangle between the lines and its 3D world counterpart (shown in the top-down view); users can gauge the correctness of the estimated camera pose using the ground plane (visualized as a grid of **yellow** dots), the **red** horizon line and the top-down view; (**2**) *top-down viewer*: the image is projected onto the ground plane and shown from a top-down perspective; the **red** rectangle is used to estimate the camera pose; (**3**) *object orientation*: the x axis represents time passing while y axis represents orientation angles $\in [-\pi, \pi)$ of the tracked object; (**4**) *manual parameters*: user-defined parameters such as metric scale of the **red** rectangle and object footprint, camera distortion parameters, smoothing coefficients for the object's trajectory points and orientations; (**5**) *object time-line*: users can use the slider to verify the quality of the tracked object's trajectory over time; the trajectory is shown in (**1**), (**2**) and along with the segmented object in the zoomed-in view below as a **green** curve; the object's footprint is highlighted in **white**.

homography **H** by placing two pairs of parallel lines on the ground plane. The

trajectory used to define a tracked object's normalized speed is formed by the center points of the input tracked 2D bounding boxes (obtained as describe in Section 4.2.1). The set of points can be smoothed using a 1D Gaussian kernel convolution and users can manually move and stretch the result to better match the object movement. Additionally, a footprint can be defined (dashed line in inset) to more accurately describe the space occupied by a car (compare to the solid line image-space bounding box in the inset). Our Counter Loop game prototype uses this information to perform collision detection. While the tool in Figure 3.10 works well in practice, we show in Sections 5.1 and 5.2 new algorithms that can perform these steps automatically, eliminating the need for users to invest their time and effort and allowing them to focus on the creative process instead.



3.2.3 Considerations

In this section we have explored ways of enabling users to interact with, traditionally passive, video content. We do this by introducing additional information alongside time progression which we can leverage to meaningfully constrain the way we reshuffle the input video frames. The visually distinctive semantic information described in Section 3.2.1 shows great promise and is at the core of our end-to-end interactive video system from Chapter 4. We will show how, despite being a very simple and intuitive abstraction, it is very flexible and easy to define interactively. We believe it represents a great step towards engaging, interactive video experiences and enables a new medium of expression that would be otherwise impossible (see Chapter 4). The speed normalization technique on the other hand, is less abstract and more tailor made to a surveillance camera scenario such as our HAVANA dataset. While we discuss, arguably prohibitive, requirements for the user, such as tracking objects through time and manually estimating camera pose, it is the reason that the Counter Loop game prototype from Section 4.6.1 has been possible. Moreover, in Chapter 5 we discuss how to ease the burden on the user and automatically recover the information needed by the speed normalization

technique.

3.3 Real-time Interaction

In the previous sections we discussed how to generate video sequences indefinitely and how to allow external factors, such as users or video game logic, to control and guide the video synthesis. These capabilities enable the creation of compelling video experiences that are always new and engaging. However, for them to be interactive and reactive to user input, we need to tackle the issue of *synthesizing new video sequences in real time*. The illusion of interacting with what's happening on the screen is in fact quickly broken if there is no instant reaction to a user's input.

In this section, we draw a parallel between video synthesis and the world of character animation. Schödl and Essa [SE02] tackle this same problem by adapting their video texture technology to allow users to control the animation of small animals such as mice. The input data to video synthesis and character animation is very different, as the first consists of images while the second typically represents a skeleton in terms of joint orientations and positions. However, they are very similar in the sense that there is time progression and the change from frame to frame is incremental. Similar to Video Textures, character animation technologies, such as [KGP02, LCR*02, AF02], reshuffle their input frames or full sequences to create something new based on some guidance. For instance, this could mean telling a character to move from a point A to a point B while running and then jumping once reaching the destination. The final animation is built by finding the correct sets of frames, showing the desired actions and concatenating them together seamlessly, just like Video Textures and what we described in Section 3.2.

3.3.1 Video Fields

In this section, we borrow concepts from Motion Fields by Lee *et al.* [LWB*10] and adapt them to video synthesis. We choose this method instead of existing alternatives as it is specifically designed for immediate response to user input,

which is critical in an interactive scenario such as what we are aiming for. The main difference to their method is that joints locations and orientations can be easily interpolated when transitioning between input frames while we currently have no way of doing the same between video frames. As such, their *continuous* "field" of motion capture frames becomes effectively a *discrete* "graph" representation in the case of our *Video Fields*.

Field set-up

Following the example of *Motion Fields* by Lee *et al.* [LWB*10], we define *states*, *actions*, *transitions* and *rewards*. Video states s are defined as a tuple $s = (f, \theta_T)$ where f is a *frame state* and θ_T are *task parameters*.

Unlike Lee *et al.* where $f = (x, v)$ is defined in terms of pose x and velocity v, we set the frame state f to the index of the frame in the original input video. We could have defined feature points (*e.g.* SIFT [Low04]) and kept track of their positions and velocities instead. However, while interpolating poses and velocities in a motion field is trivial, image interpolation is still actively researched [SLWSS15]. Here, we are most interested in the real-time control features of the field and thus focus on them.

The *task parameters* θ_T keep track of how well a task is performed and we define it as a combination of the following two:

1. show frame states belonging to a certain semantic action as defined in Sections 3.2 and 4.2.3, meaning $\theta_T = [l_i, i \in [0, C)]$ where l_i is the probability of the frame state f of showing label i of the C user-defined action classes

2. show a randomly selected frame f within a certain semantic class, meaning $\theta_T = f$

To traverse a Video Field $\mathcal{F}$ starting from state s, we choose one *action* $a \in \mathcal{F}(s)$ out of k predefined ones. As seen in Lee *et al.* [LWB*10], we define each action a to favor one of the k neighbors of s as

$$a = \{\frac{a_i}{\|a_i\|} | a_i = w_0, ..., w_{i-1}, 1, w_{i+1}, ..., w_{k-1}]\}, \tag{3.12}$$

where w_i is a similarity weight indicating how similar the frame state f is to its

i^{th} neighbor f_i. We define $w_i = \frac{1}{\eta}\frac{1}{d(f,f_i)^2}$ where η normalizes the vector of weights to sum up to 1 and $d(f, f_i)$ is some measure of similarity.

Given state s and action a, we use the integration function $I_s(s, a) = s'$ to transition to state $s' = (I(f, a), \theta'_T)$, where $I(f, a) = f'$ returns the neighbor f' of f favored by a and θ'_T are the task parameters of f'.

Finally, we define a reward function

$$R(s, a) = \lambda R_T(s, a) + (1 - \lambda)R_A(s, a), \tag{3.13}$$

where λ balances the task versus appearance reward contributions and we set it to 0.7. The task reward $R_T(s, a) \in [0, 1]$ is defined as

$$R_T(s, a) = \frac{2 - |\theta_T - \theta_T^d|}{2} \tag{3.14}$$

with θ_T^d denoting the desired task parameters while the appearance reward $R_A \in [0, 1]$ is

$$R_A(s, a) = \frac{p(f, f_i)}{\max_{f_i \in \mathcal{N}}[p(f, f_i)]}, \tag{3.15}$$

where $p(f, f_i)$ denotes the probability of going to the neighboring frame state f_i from f and is defined as a zero-mean Gaussian based on the similarity measure defined in [SSSE00].

Action choice policy

A new video is synthesized by traversing the video field. An action $a \in \mathcal{F}(s)$ associated to the current state s is chosen which results in moving to the neighbor s' it favors. A trivial way to choose an action is by selecting the one that gives the highest reward $R(s, a)$. Lee *et al.* call this the *greedy choice policy*

$$\pi_G(s) = \operatorname*{argmax}_{a \in \mathcal{F}(s)}[R(s, a)]. \tag{3.16}$$

Unfortunately, as Schödl *et al.* [SSSE00] point out, a greedy policy cannot ensure that traversal will not reach dead-ends, *i.e.* regions where there are little or no good transitions between other parts of the video. Moreover, the greedy policy

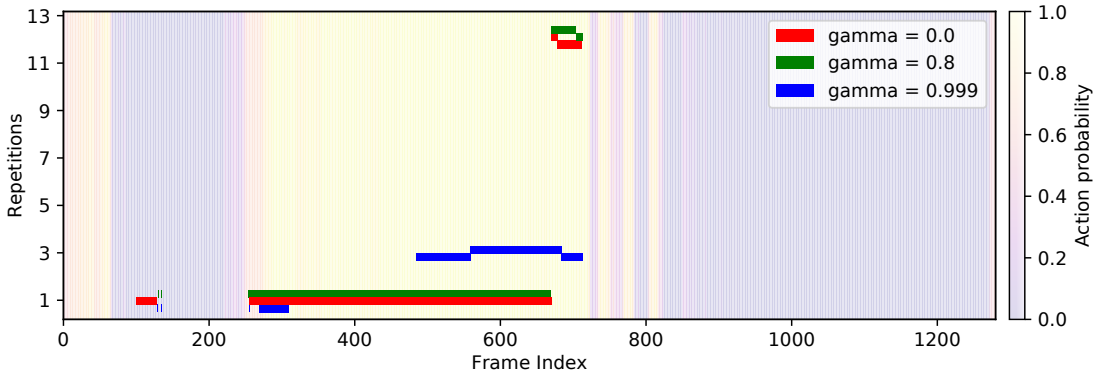


Figure 3.11: We visualize the effect γ has on looping videos. We run video fields starting from frame 100 and request a semantic action. Yellow frames have higher probability of showing this desired action, *i.e.* they should be shown more often as they yield higher task rewards. With low values of γ, focus is mostly on immediate reward and the traversal follows the input time-line to get stuck in a short loop. If γ is set to a high value, the traversal takes into account future reward which results in jumping across the time-line more often to reach longer loops.

does not guarantee we will ever reach video states that agree with the desired task parameters θ_T^d. The solution, first proposed in Video Textures and then adapted by Lee *et al.*, is to estimate the future reward that a certain chosen neighbor leads to (which will be low if it leads to a dead end). We define the *lookahead policy* as

$$\pi_L(s) = \underset{a \in \mathcal{F}(s)}{\operatorname{argmax}} \left[R(s,a) + \max_{\{a_t\}} \sum_{t=1}^{\infty} \gamma R(s_t, a_t) \right], \tag{3.17}$$

where the right-hand side of argmax[·] takes into account the reward given by the future choices if action a is chosen and γ balances how much we focus on short versus long term rewards. Figure 3.11 shows the effect of γ on looping. By putting more emphasis on long term rewards (*i.e.* γ is closer to 1), we avoid following the time-line and getting stuck in short loops. Instead, the algorithm is willing to jump across the input time-line to reach better, longer loops.

As computing all possible rewards from a on is infeasible, we use Q-learning [Wat89]

and redefine π_L in terms of a *value function* $V(s)$

$$\pi_L(s) = \underset{a \in \mathcal{F}(s)}{\operatorname{argmax}} \left[R(s,a) + V(I_s(s,a)) \right], \tag{3.18}$$

where $I_s(s,a) = s'$ is the integration function as before and the value function is

$$V(s') = \max_{a \in \mathcal{F}(s')} \sum_{t=1}^{\infty} \gamma R(s_t, a_t). \tag{3.19}$$

We can then define $V(s')$ recursively in terms of the value of $V(s)$ at other video states s such as for instance

$$V(s_i) = R(s_i, \pi_L(s_i)) + V(I_s(s_i, \pi_L(s_i))), \tag{3.20}$$

where the value function at video state s_i depends on the reward of taking action $\pi_L(s_i)$ and the value of $V(s)$ at the video state we reach by taking that action. This allows us to use Q-learning and iteratively update the value of $V(s)$ and $\pi_L(s)$ until convergence. Following the example of Lee *et al.* we initialize $V(s_i) = 0$ for each video state s_i in the field and optimize a different version of $V(s)$ for each possible set of task parameters θ_T^d.

Field traversal

Once we define a video field $\mathcal{F}$ and an action choice policy $\pi_L(s)$ we can create and control a video texture in real time. We choose a starting video state s and desired task parameters θ_T^d and at each time step we use Equation 3.18 to choose the best action a and the integration function $I_s(s,a)$ to get the next video state. Figure 3.12 shows the effect γ has on the traversal. Higher values, meaning there is more focus on long term rewards, results in the ability to switch between semantic actions faster with little to no penalty in visual fidelity (*i.e.* the amount of appearance reward is comparable with lower values of γ). Moreover, there is more variability as more of the input frames are shown and the desired semantic action is shown for longer. We show a result video demonstrating meaningful real time control of video synthesis on our website[1].

[1] https://corneliu.co.uk/phdresults/chapter3/index.html#ribboncontrol

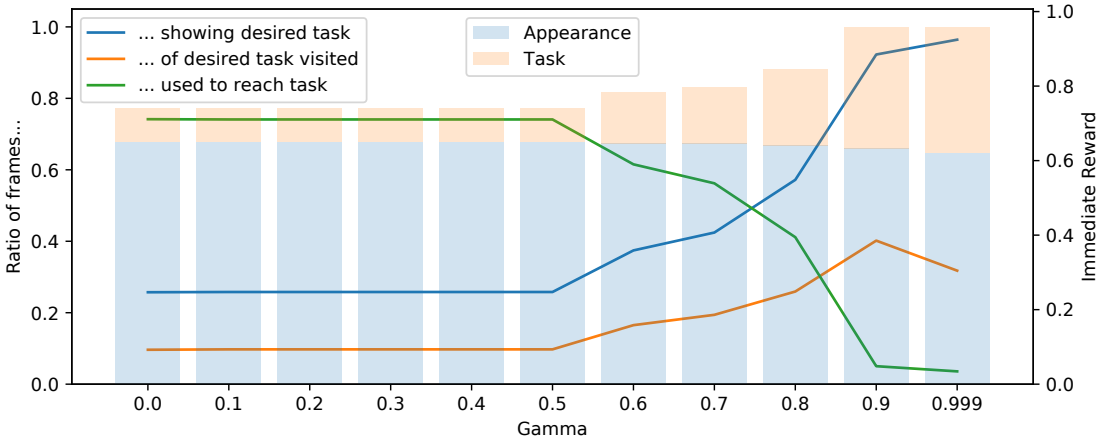


Figure 3.12: We visualize the effect γ has on traversing the video field for 50000 frames. The **blue** line represents *accuracy* as it shows how often a chosen frame is showing the desired semantic action (higher is better). The **green** line represents *speed* as it shows how many frames are necessary to reach frames showing the desired task (lower is better). The **orange** line represents *exhaustiveness* by counting how many frames showing the desired semantic action have been visited during looping (higher is better). The bar graphs indicate the average immediate reward yielded by each shown frame during looping (higher is better). More focus on long term rewards, *i.e.* higher values of γ, results in higher accuracy, speed and exhaustiveness. Higher values of γ also result in much larger task rewards at the expense of slightly smaller appearance rewards.

3.3.2 Considerations

In this section, we have made the point that for a successful interactive video experience, the algorithm in charge of synthesizing frames by, for instance, reshuffling the input ones, should perform at real time rates. This is necessary because users need to immediately receive feedback regarding their input or the illusion of interacting with the video and that there is consequence to their action is broken. The graph-based method described in Section 3.3.1 draws a parallel to character animation systems and, as such, is able to react to user input instantly. This allows us to control the direction in which the RIBBON from Figure 3.7 flutters. There is direct reaction to the user changing the wind direction, making for an engaging experience very dissimilar from simply watching the input video[1]. However, the system we describe in Chapter 4 highlights (and addresses) the limitations of

[1] https://corneliu.co.uk/phdresults/chapter3/index.html#ribbon

such a graph-based approach. First, there is no straightforward extension to enable controlling multiple objects at the same time (*e.g.* multiple candle flames in Figures 3.1b and 4.12a). Second, even if we could somehow extend the method to support multiple objects, there would be no way to ensure visual compatibility between separate objects, such as guaranteeing that all candles in Figure 4.12a flicker in the same direction (see Section 4.3.1 for more details). Finally, even with the lookahead action choice policy in Equation 3.17, we have still experienced dead-ends using our RIBBON dataset, likely due to the imperfect similarity measure (please see our result video online[1]).

3.4 Foreground Segmentation

In the previous sections, we have discussed three of the ingredients necessary to creating compelling and interactive video experiences. Key to introducing them into the mix is the ability to meaningfully reorder the input video frames. However, a simple reshuffle drastically limits the possible outputs because of two reasons. First, videos often contain *multiple* dynamic elements performing different actions at the same time so they would be forced to interact with each other or move at the same time as they did when originally filmed. Second, even if the video only contained one dynamic object, we would be forced to always show it moving against the same background and there would be no way to convincingly place it in a different environment such as for instance a video game level (see flag in Figure 3.17).

To overcome these limitations, we need to be able to reason about movement at single or groups of pixels as opposed to at the full video frame level, a process called *segmentation*. For instance, pixels can be grouped together into patches based on how similar their colors are (a.k.a. super pixels such as [ASS*12]), whether they belong to the same object (a.k.a. semantic segmentation such as [LSD15]) or even whether they are considered to be foreground or background (*e.g.* [Pic04]). The choice of segmentation depends largely on the application. For instance, in Figure 3.15, we could perform foreground-background segmentation to separate the fluttering flag from the rest of the scene or we could do object segmentation

[1] https://corneliu.co.uk/phdresults/chapter3/index.html#ribboncontrol

to separate each moving element (such as the flag and the people walking by) from one another and from the background.

In this Section, we briefly explore two methods for performing *accurate* (as opposed to the *approximate* results we aim for in Chapter 4) foreground-background segmentation. They were both instrumental in enabling two video game prototypes that demonstrate how videos can be transformed into interactive assets usable from within a game engine. The differences between them are due to the nature of the input video. Therefore, in Section 3.4.1 we discuss methodology for segmenting opaque objects such as the flag in Figure 3.17 and in our first game prototype[1] where a *binary* mask is enough. In contrast, in Section 3.4.2 we show how to perform *alpha matting* of semi-transparent objects such as the flame in Figure 3.1b and our second game prototype[2], where pixels are assigned a *continuous* value of "foregroundiness".

3.4.1 Example-based Segmentation

The first foreground segmentation algorithm we experimented with learns a model of appearance from user-given examples. We have chosen it for its simplicity, speed and the fact that users can quickly improve results if they are willing to invest the time and effort.

First, users use our tool, shown in Figure 3.13, to manually segment a few frames. Instead of asking for a full frame mask, which would require considerably more effort, we allow users to scribble over foreground pixels with a **red** brush and over background pixels with a **blue** brush (as shown for the example frame in Fig. 3.13). We automatically propagate the scribbles to the remaining pixels in the same frame using the *Watershed* algorithm [Mey92]. The Canny edge detector [Can86] is used to find the transition between foreground and background areas, which are then inflated by a user-defined number of pixels. This results into the *trimap* shown in Figure 3.13 where **blue** pixels are definitely background, **red** pixels are definitely foreground and **green** pixels are undefined.

Traditionally, optical flow-based methods are used to propagate the user

[1] https://corneliu.co.uk/phdresults/chapter3/index.html#arrowgame
[2] https://corneliu.co.uk/phdresults/chapter3/index.html#candlegame

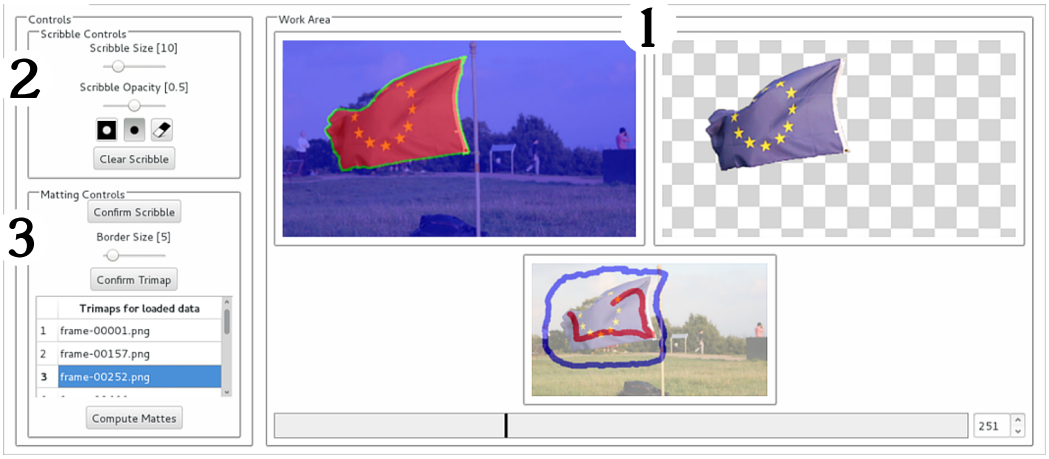


Figure 3.13: Example-based segmentation tool. (**1**) *work area*: users can scribble onto a frame (middle bottom) using various brushes to define a trimap (shown top left) that is used to train an RFR to regress the segmentation mask for each frame (shown top right); (**2**) *scribble controls*: users can select the type of brush to scribble with (**red** for foreground and **blue** for background) or whether to erase parts of or all scribbles; (**3**) *matting controls*: shows which frames the user defined a trimap for which can be selected and modified as desired.

annotations to the remaining video frames (*e.g.* Chuang *et al.* [CAC*02] propagate the trimaps while Lang *et al.* [LWA*12] directly propagate the scribbles). Instead, we use the definitely foreground and background pixels to learn a model of appearance of the foreground object. The model we have chosen is a Random Forest Regressor (RFR) [Bre01] which is trained to distinguish foreground pixels from background ones based on their three-channel color and image-space coordinates. As the object changes appearance or position drastically, the RFR predictions become noisy. However, users can use our intuitive tool (see Fig. 3.13 and our supplemental website[1]) to quickly scribble over bad frames and correct mistakes by adding them to the pool of learning examples used to train the RFR.

3.4.2 Intensity-based Segmentation

The second segmentation technique we experimented with, was specifically designed to deal with the use case of filming light emitting elements such as the

[1] https://corneliu.co.uk/phdresults/chapter3/index.html#flagsegment

flame in our CANDLE dataset (see left image in Fig. 3.14). Provided they are filmed in a dark environment against a black background, the pixel intensity values can be used as a proxy for the alpha matte separating the flame from the rest of the scene. We use the standard RGB-to-Gray formula to define the alpha values $\mathtt{A} = 0.21\mathtt{R} + 0.72\mathtt{G} + 0.07\mathtt{B}$. While we could use A directly to create the flame asset we used in our game prototype shown in Figure 3.14, we have found that small tweaks can improve quality drastically For this reason, we built the tool shown in

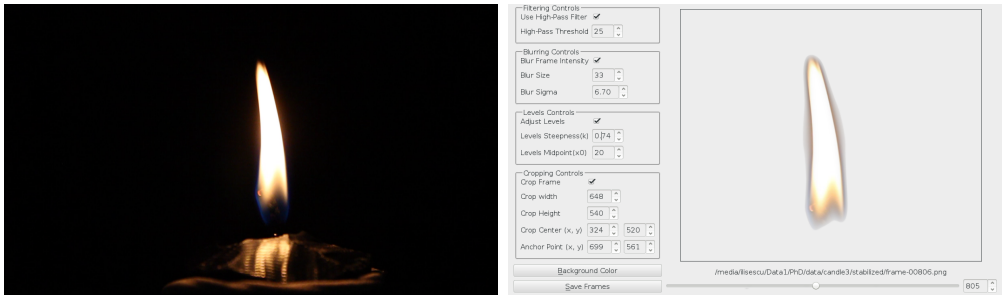


Figure 3.14: An input video such as the the CANDLE dataset (left) can be segmented and cropped using our intensity-based segmentation tool (right). The controls visible on the left side of the tool can be used to manipulate the final result as described in the text.

Figure 3.14, which users can use to adjust the alpha matte in three ways. First, a high-pass filter can remove undesirable highlights such as seen on the candle in Figure 3.14. Second, blurring $\mathtt{A}$ using a Gaussian filter can improve the quality of the transition between foreground and background. Finally, passing the alpha values through a sigmoid function can further tweak the appearance of the flame. If necessary, users can also crop and recenter the segmented video (compare the input frame on the left to the output frame on the right in Figure 3.14) to prepare it for usage from within a game engine as discussed in Section 3.5.1. Please see the controllable candle flame game asset on our supplemental website[1].

3.4.3 Considerations

We have dedicated the above sections to exploring video segmentation in the context of enabling new video experiences. The RFR-based method is the most

[1] https://corneliu.co.uk/phdresults/chapter3/index.html#candlegame

general of the two and it performs well for our use case. It does however make a set of assumptions such as the fact that the object is mostly stationary, has distinctive appearance w.r.t. the background and users are willing to put effort into annotating examples if the results are not satisfactory. The intensity-based method on the other hand, while less demanding in terms of user effort, it is tailor made for our use case and would likely not generalize well to other videos. However, both methods have proven instrumental in enabling the creation of the two game prototypes shown in Figures 3.1b and 3.17 and our website[1]. Moreover, they highlighted issues about generalization and required user effort which we address in later chapters (*e.g.* Section 4.2.2 and Chapter 5).

3.5 Video Authoring

In the previous sections, we have described the properties that a compelling and engaging interactive video experience should have along with methods that make such properties a reality. We have also shown that it is often desirable to involve users, both for assisting imperfect automatic algorithms (*e.g.* segmentation in Sec. 3.4) and providing creative input (*e.g.* Section 3.2.1). It stands to reason then, that it is crucial to design effective tools that facilitate the *human-computer interactions* needed to both support content *creators* in making video experiences and enable *consumers* to enjoy them.

The field of Human-Computer Interaction (HCI) raises very important questions about how we interact with machines. We believe we must strive to build tools that are enabling, easy to learn but most importantly cohesive and self contained by providing all the needed functionality. In Chapter 4 we will describe such a tool in detail and carefully analyze its HCI merits. For completeness, we now briefly describe another tool which was instrumental in demonstrating how videos can be processed and integrated within traditional video game engines adding extra detail to the experience (please see our game prototype video[2]). We believe it shows how clever algorithms and intuitive user interactions can come together in an end-to-end setting to enable new ways of enjoying video content.

[1] https://corneliu.co.uk/phdresults/chapter3/index.html
[2] https://corneliu.co.uk/phdresults/chapter3/index.html#arrowgame

3.5.1 Creating Video Textures for Video Games

The tool we now describe gives content creators the ability to take an arbitrary video, such as the flag in Figure 3.13, segment it from its original background as shown in Section 3.4.1, loop it indefinitely using the Video Textures technique [SSSE00] and easily export the result as a video asset to the Unity3D game engine. While we discuss the segmentation capabilities of the tool and show them in Figure 3.13, we here present and show in Figure 3.15, the more creative side of

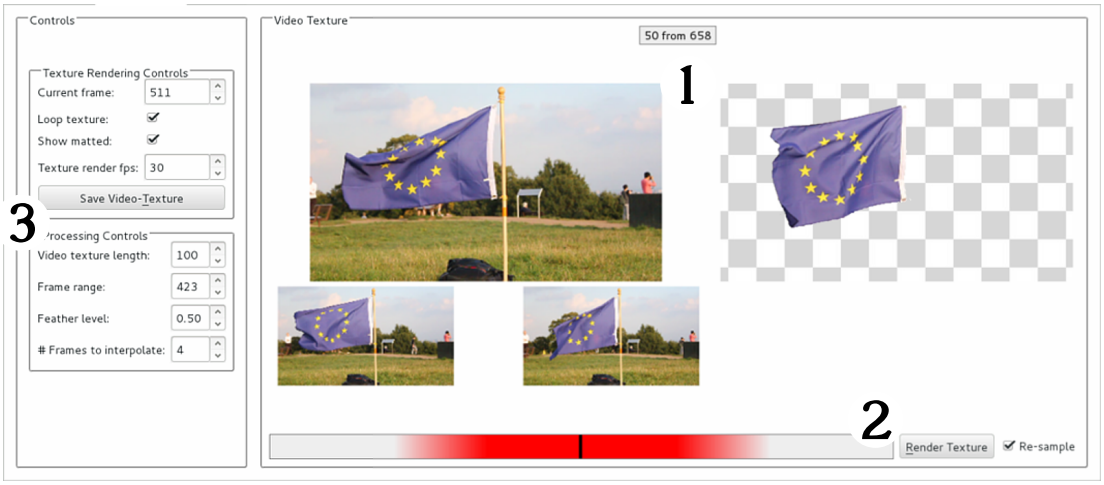


Figure 3.15: Video textures creation tool. (**1**) *main viewer*: the left hand side shows the currently selected anchor frame S and the start and end frames of the frame interval highlighted in **red** in (**2**); the right hand side shows the current looping video texture; (**2**) *video time-line*: the anchor frame is highlighted by a **black** bar while the subset of input frames to loop through is highlighted in **red**; (**3**) *user controls*: manually defined parameters such as the length of the video texture and the frame subset size.

the process. A video showcasing a typical session using our tool can be found on our website[1].

Video Textures

For completeness, we first describe how video textures [SSSE00] work. As we briefly mentioned in Section 3.1, the main idea behind seamlessly playing back a finite length video for an indefinite amount of time is to find similarly looking

[1] https://corneliu.co.uk/phdresults/chapter3/index.html#flagauthor

frames that can be used interchangeably and are thus suitable locations for jumping around the original video's time-line. To do so, we define an $N \times N$ *distance matrix* $\mathbf{D}$. Each element (i, j) in $\mathbf{D}$ represents the similarity between frames $\mathtt{I}_i$ and $\mathtt{I}_j$ and is defined as

$$D_{ij} = d_E(\mathtt{A}_i\mathtt{I}_i,\ \mathtt{A}_j\mathtt{I}_j), \tag{3.21}$$

where d_E is the euclidean distance in Equation 3.1 and $\mathtt{A}_i$ and $\mathtt{A}_j$ are alpha mattes associated to each frame as defined in Section 3.4.

Schödl *et al.* [SSSE00] note that preserving dynamics is crucial when jumping to different locations in the input video. This is best explained with the example in Figure 3.16, where both frames pointed to by the **red** and **blue** arrows are very

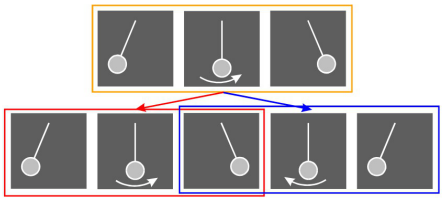


Figure 3.16: The importance of preserving dynamics. While the middle **yellow** frame is similar to both the middle **red** and **blue** ones, only jumping between the **yellow** and **red** frames preserves the correct movement of the pendulum.

similar to the central **yellow** frame. If we were to use the **yellow** frame interchangeably with the **blue** one however, the result would be the clearly mistaken sequence of frames highlighted in **blue**. To avoid such situations and ensure the **red** frame is always picked, Schödl *et al.* suggest taking into consideration the similarity between neighborhoods when defining the distance between two frames i and j. They define the new distance $\mathbf{D}'$ such that

$$D'_{ij} = \sum_{k=-m}^{m-1} w_k D_{i+k,\, j+k}, \tag{3.22}$$

where m denotes the size of the neighborhood and w_k weighs the importance of each neighboring frame. In Figure 3.16, k would be set to 1 and it becomes clear that the new distance measure correctly identifies the **yellow** set of frames as being more similar to the **red** sequence than the **blue** one.

The distance $\mathbf{D}'$ can now be used to find similar looking frames (*i.e.* low

distance) and randomly use them interchangeably to jump to different locations of the input video. However, doing so may lead us to unique parts of the input video where there are no seamless jumps. Therefore, we would be forced to use badly visible jumps and breaking the illusion of watching an infinitely long video. In order to predict whether a good jump leads to a dead ended section of the input video, Schödl *et al.* [SSSE00] propose a very elegant solution. They manipulate $\mathbf{D}'$ to associate higher values to seemingly low cost jumps if they are likely to lead to dead ends. To this purpose, we define $\mathbf{D}''$ with

$$D''_{ij} = (D'_{ij})^p + \alpha \min_k D''_{jk}, \tag{3.23}$$

which intuitively means that the distance between frames i and j depends on how similar their appearance is but also how good the jumps will be in the future if i and j are used interchangeably. Here, p controls the trade off between using multiple low cost jumps (*i.e.* similar pairs of frames) and using a single high cost one. To compute $\mathbf{D}''$, we follow [SSSE00] and define $m_j = \min_k D''_{jk}$, initialize it using the values of $\mathbf{D}'$ and iteratively update the values of m_j and $\mathbf{D}''$ until convergence (a process known as Q-learning [Wat89]).

Loop finding

To create a video texture, we use the random choice-based method described in [SSSE00] which consists of randomly showing frame j after frame i based on the probability

$$P_{ij} = \exp - \frac{D''_{i+1,\,j}}{\sigma} \tag{3.24}$$

where σ controls how much we are willing to tolerate bad jumps. The tool in Figure 3.15 also allows users to select a subset of the input frames to favor during looping. They manually choose an anchor frame S and a neighborhood size (visually marked by the black bar and red highlight in the time-line slider at the bottom of Figure 3.15 respectively). The probability of showing frame j after frame i then becomes

$$P_{ij} = \exp - \left[\frac{D''_{i+1,\,j}}{\sigma} + f_S(|j - S|) \right] \tag{3.25}$$

where f_S is the sigmoid-like smooth step function. Users have manual control over the steepness of f_S which we indicate visually by feathering the red highlight in Figure 3.15. Intuitively, the further a frame j is from the anchor frame S, the more unlikely it is to be used despite it potentially producing a seamless transition from frame i.

Unity3D integration

Using the loop finding technique described above, we can generate an arbitrarily long sequence of frames with no visible transitions. In order to integrate such a seemingly infinite video into a game engine such as Unity3D, we have chosen a well known technique known as billboarding. It consists of placing two-dimensional rectangular meshes in a three-dimensional virtual scene and actively changing their orientation in response to the user-controlled camera. This method was originally designed as a way to represent complex 3D geometry such as entire tree branches with leaves by means of appropriately placed and textured simple planes. It is perfect for our use case, as we have filmed complex animated geometry such as our FLAG dataset and can approximate it with a simple rectangular billboard which effectively acts as a television screen. The tool presented in this section can automatically export the necessary Unity3D assets which can then be carefully placed within a 3D game level together with additional geometry such as a flag pole, resulting in the convincingly animated flag shown in Figure 3.17 and our supplemental video[1].

3.5.2 Considerations

The section above describes a tool that assists content creators in producing video textures [SSSE00] and export them as game assets usable from within the Unity3D engine. We believe that having an end-to-end tool is crucial to enabling interactive video experiences because of two reasons. First, no matter how clever automatic algorithms are, such as the video looping technique by Schödl *et al.* [SSSE00], their performance will depend upon the input data. We believe that, having the ability to involve the user is critical for achieving the best results. Second, in

[1] https://corneliu.co.uk/phdresults/chapter3/index.html#arrowgame

Figure 3.17: Example of final video texture used in a game level in the Unity 3D engine. The flag is derived from real video footage whereas the remaining elements are traditional assets (left) or IBR assets (right).

the context of creative experiences such as the ones described in Chapter 4, it is always important to give the content creator the ability to express themselves and influence the output. As we will further discuss in the next chapters, this is one of the reasons that motivated our choices throughout this thesis.

3.6 Multiview Interaction

In the previous section, we have discussed a simple way of showing traditionally two-dimensional video content in a 3D video game world by means of flat billboards. As visible in Figure 3.17, the effect can be quite convincing, provided we do not stray away from the input view too much and the filmed object is roughly planar. As soon as either of these prerequisites is not satisfied by, for instance, looking at the flag billboard from a grazing angle, the illusion breaks. Our goal is to enable the creation of new interactive video experiences that immerse and engage users so maintaining the illusion at all times is crucial.

So far in this Chapter, we have defined videos as sequences of flat grids of pixels and manipulated them as such. However, a more accurate way of looking at videos is as a sequence of projections of a three-dimensional world onto a 2D image plane. If we can reason directly about the 3D world, we can have a more accurate representation of the movement captured by the video and, as a result, design more natural and intuitive interactions. Take the example of the car from our HAVANA dataset moving across the road crossing in Figure 3.18: if we think

of the video in terms of a sequence of images, all we can do convincingly is to manipulate the way we play back the frames and manipulate the speed at which the car is moving (as discussed in Section 3.2.2). Crucially, we could not for instance, make the car follow a different path convincingly as we would not know how it looks from a different point of view or how its size changes as it moves closer to or further away from the camera.

There are two reasons we cannot yet enable such natural and intuitive interactions: i) we do not know how an object moves through the scene and how consequently its appearance changes and ii) we do not have any three-dimensional information about the world and how it is projected onto the 2D video frames and as such we cannot synthesize new visuals by leveraging this more natural representation. To tackle the above issues we need ways of inferring additional information about the world visible through the video frames and ways of leveraging such information to create 3D visuals by manipulating two-dimensional photos. We dedicate Chapter 5 to exploring automatic methods for inferring the extra information we need, while we discuss our experimental way of leveraging it for rendering 3D visuals in the next Section 3.6.1.

3.6.1 Generating 3D Visuals from 2D Video

In this Section, we present a *real-time* 3D viewer that leverages video data to create its visuals. We assume the input videos are captured using a static calibrated camera with known pose (*e.g.* by using the tool from Section 3.2.2) and that filmed objects are tracked over time and their position in the 3D world is know at each step. We discuss how to automatically infer this information in Chapter 5.

Our 3D viewer shown in Figure 3.18 and on our supplemental website[1] is implemented using OpenGL which defines a virtual camera in terms of a *view* matrix $\mathbf{V}$ and a *camera projection* matrix $\mathbf{C}$. In contrast, our calibrated camera is defined in terms of its internal parameter matrix $\mathbf{K}$ and camera pose $\mathbf{P} = \begin{bmatrix} \boldsymbol{\Omega} & \boldsymbol{\tau} \end{bmatrix}$ as detailed in Section 5.1. To simulate such camera using the OpenGL conventions

[1] https://corneliu.co.uk/phdresults/chapter3/index.html#3dvis

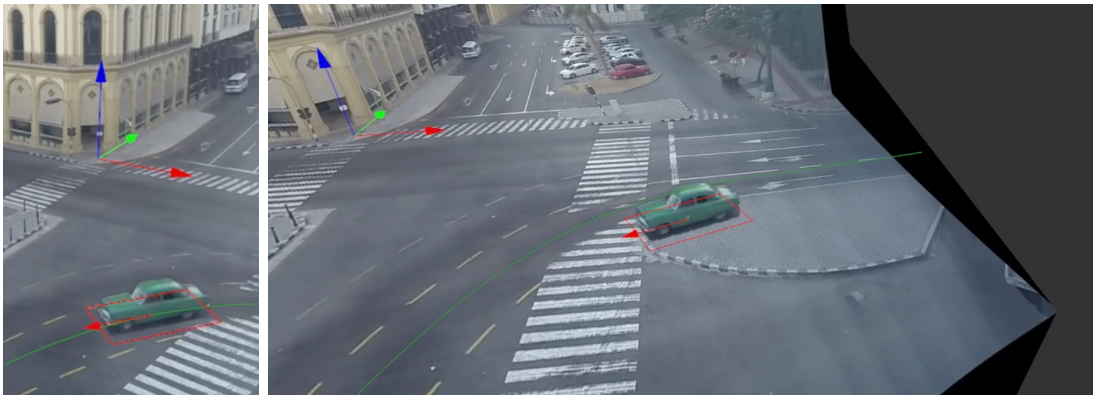


Figure 3.18: Left, we show a cropped input video frame of a tracked object. Right, we show the same object rendered in our 3D visualizer from a novel view point. Note how the car still looks realistic despite looking at it from a never-before seen viewpoint.

we set

$$\mathbf{V} = \begin{bmatrix} \boldsymbol{\Omega} & \boldsymbol{\tau} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.26}$$

where $\boldsymbol{\Omega}$ is the camera rotation and $\boldsymbol{\tau}$ is the column vector representing the camera's center of projection. The camera projection matrix

$$\mathbf{C} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \phi_x & \gamma & \delta_x & 0 \\ 0 & \phi_y & \delta_y & 0 \\ 0 & 0 & n+f & nf \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.27}$$

where ϕ_x, ϕ_y (x and y *focal lengths*), γ (*skew*) δ_x and δ_y (x and y image center *offsets*) are the internal camera parameters defined in the intrinsics matrix

$$\mathbf{K} = \begin{bmatrix} \phi_x & \gamma & \delta_x \\ 0 & \phi_y & \delta_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.28}$$

and n and f are the near and far clipping planes respectively. The left-most

matrix on the right-hand side of Eq. 3.27 maps the camera space points (*i.e.* after they are multiplied to the view matrix) to the $\left[-1, 1\right]$ interval so we set $l = 0$, $r =$ width, $b =$ height and $t = 0$ according to the viewport size.

Rendering filmed objects

After setting up the OpenGL environment as described above, we are ready to render the filmed scene. The static background is rendered by projective-texturing the approximate scene mesh we manually define using the tool described in Section 5.1.1. Filmed objects such as the car in Figure 3.18 are rendered by placing flat billboards aligned to the tracked footprint we are given as an input (as showcased in our supplemental video[1]) and texturing them using the patches from the input video segmented from the static background by, for instance, using the interactive method described in Section 4.2.2. Users can manually control a filmed object and place it in the 3D scene as they please, so we choose the patch from the video frame where the object is seen from the most similar viewpoint to the one expected by the user given the pose of the virtual camera and the location of the object. We do this automatically by comparing the user-defined object pose w.r.t. the virtual camera to its pose w.r.t. the camera that captured the input video at each frame and picking the best match. Clearly, if the user chooses to place the car on its original trajectory and the virtual camera where the camera was when capturing the input frame sequence (*i.e.* using Equations 3.27 and 3.26), the result would replicate the input video faithfully.

In many ways, the technique we describe above is similar to *Unstructured Lumigraph Rendering* (ULR) [BBM*01] with notable differences being the simpler shape proxy (*i.e.* the flat billboard) and rendering a chosen patch as a whole as opposed to selecting the best matching patch per pixel. We have chosen this path because results would likely be similar given that we represent each object with such simple geometry and drastic changes in pose and location would result in poor results either way. However, we explore better geometry proxies in Chapter 5 and discuss extending our viewer to ULR in Section 5.3.2.

[1] https://corneliu.co.uk/phdresults/chapter3/index.html#3dvis

3.6.2 Considerations

We have presented a real time 3D viewer which can leverage knowledge about the real world (*e.g.* the camera pose) to manipulate 2D video frames in meaningful ways. We make a set of critical assumptions about the scenes and objects we are dealing with, such as having a static camera and a roughly planar ground surface. Despite these, we believe it sets the basis to more natural interaction with video content and brings us closer to our goal of making the act of watching a video more game-like[1]. While reasoning about the 3D world is not a prerequisite to interactivity (as demonstrated by our new medium of expression in Chapter 4), it can lead to more immersive experiences as witnessed by the fact that the most successful modern video games emulate the three-dimensionality of the real world.

3.7 Conclusions

In this chapter we sought out to identify the properties that a successful interactive video experience should exhibit. In particular, we were interested in clearly defining desirable ways of interacting with video content that is traditionally consumed passively and what making them a reality would entail. Through extensive discussions with actively involved game developers, we have found that, in order to make the activity of watching a video more game-like, we must have the ability to synthesize visuals indefinitely from a finite length sequence of frames and that we must give content creators and players the ability to influence the synthesis process in meaningful ways. In addition to these two main abilities, game developers also pointed out that they need the synthesis process to be able to instantly react to user input and that they want the video content to live in a three-dimensional world (as opposed to the 2D video frames) that most modern video games emulate. Finally, it became apparent that, as they are content creators, game developers need to be able to express their creativity so automatic algorithms to make the above a reality are just one side of the coin. In fact, we find that it is crucial to support users with automatic algorithms rather than

[1]See our result video at https://corneliu.co.uk/phdresults/chapter3/index.html#3dvis

replace them, so designing intuitive and responsive human-computer interactions becomes crucial.

In addition to defining the above desirable properties of interactive video experiences, we set out to identify and explore techniques that enable them. For instance, we believe looping is the best way to create visuals from video indefinitely, so in Section 3.1 we discuss how to do this and explore one of the most pressing issues, *i.e.* choosing a measure of similarity that can robustly identify seamless jumps within the input video's time-line. We find that semi-supervised perceptual measures such as the Random Forest Regressor in Section 3.1.2 are better suited than traditional objective distances (Sec. 3.1.1). However, they require additional user effort making them undesirable in certain cases, such as shown in Chapter 4, where we choose to focus manual intervention elsewhere (*i.e.* defining actions in Section 4.2.3). We introduce the concept of semantic looping in Section 3.2.1 which will allow us in Chapter 4 to enable users to interact with engaging video experiences. We also experiment with a character animation technique [LWB*10] which we adapted to synthesize video content in real time. While promising, we have found it was not flexible enough for our purposes so will introduce an alternative in Section 4.3.2. Further, we identified segmentation (Sec. 3.4) as indispensable for integrating video content into game engines such as Unity3D (Sec. 3.5) and experimented with a number of techniques. While reasonably successful for our practical needs, they are unlikely to generalize well to other use cases so they were not developed further. Finally, we have shown that 2D video content can be leveraged to generate three-dimensional visuals akin to modern video games. The prototype 3D viewer presented in Section 3.6.1 is promising and is further developed and extended in Chapter 5.

In Chapter 4, we take some of the most promising techniques such as the semantic looping from Section 3.2.1 and the most pressing requirements such as real time video synthesis (Sec. 3.3) and effective authoring tools (Sec 3.5) and combine them in an end-to-end system that enables a novel medium of expression. Content creators are given the tools to efficiently prepare video content and users the ability to directly interact with the resulting video experience which reacts to their inputs instantly.

Chapter 4

Responsive Action-based Video Synthesis

In the previous chapter, we have presented a number of features that we have identified, together with game developers, as crucial for creating interactive experiences based on video content. Some are more obvious, such as the ability to loop a video indefinitely in order to not put a time limit on how long consumers can enjoy the experience. Some are, perhaps, less straight-forward such as devising simple and intuitive ways that can engage players by giving them power over the output, while at the same time are powerful and generic to cater for a wide-variety of input videos. In this chapter, we take ideas from Chapter 3 and adapt and integrate them into an *end-to-end system*, which as discussed in Section 3.5 is important to ensure the efficiency of content creators.

Given our initial hypothesis, we design our system to allow and encourage content creators to improve upon and interactively change automatically inferred information (*e.g.* tracking and segmentation in Fig. 4.3). Crucially, our interactive system enables a new *responsive medium of expression* by facilitating the creation of compelling interactive experiences that engage audiences with video content that instantly reacts to their actions. As Dan Olsen outlined in his SIGCHI acceptance speech [Ols12], a great medium of expression is characterized by three properties: Range, Empowerment, and a Balanced Structure. Good *range* indicates a wide variety of possible expressions, *empowering* mediums lower the required skills and

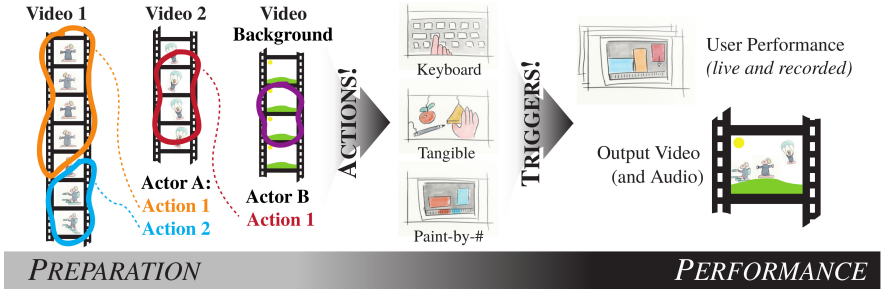


Figure 4.1: Illustration of how content creators prepare videos into liver performance for audiences to interact with. Our end-to-end technology assists with finding and segmenting loopable actions in video inputs (**orange**, **blue**, **red**). Then, discrete but compatible actions can easily be triggered during a live show.

cost to reach excellent results while a *balanced structure* constrains the user to make new outputs possible. By these measures, we find Live Looping [Pet16], where music is recorded and played back in real-time, to be an inspirational medium for authoring music. Present-day musicians like Reggie Watts[1] and Kimbra[2] can easily accumulate simple sounds, faithful to the original audio-clips, yet they have the flexibility and precise control to overlay and repeat clips to compose complex music that transcends their solo-musician appearance. In this chapter, we describe a novel end-to-end system that enables the existence of a "cousin" of Live Looping for the video domain[3], as illustrated in Fig. 4.1.

Presently, technologies for video-authoring have good Range [Ols12], meaning that they are flexible and accurate in depicting many subjects. But they lack a Balanced Structure and Empowerment, which require confining flexibility to ensure even novices succeed, without curtailing what experts can create. Our goal is to develop a tool that enables end-users to express themselves through a new medium characterized by all of these three properties. In particular, we aim to i) "lower the floor" so that novices can participate, ii) "raise the ceiling" so that a single artist can compose expressive pieces and performances while iii) catering for the widest range of inputs possible. We do this by building an *end-to-end system* to create, iteratively repair, and control video sprites thanks to

[1] https://youtu.be/0gKWfvd-chA?t=123s
[2] https://youtu.be/DgmoHtnoi7k?t=27
[3] YouTube's MysteryGuitarMan uses labor-intensive methods and has almost 3M followers: https://youtu.be/EQXA7ErL708

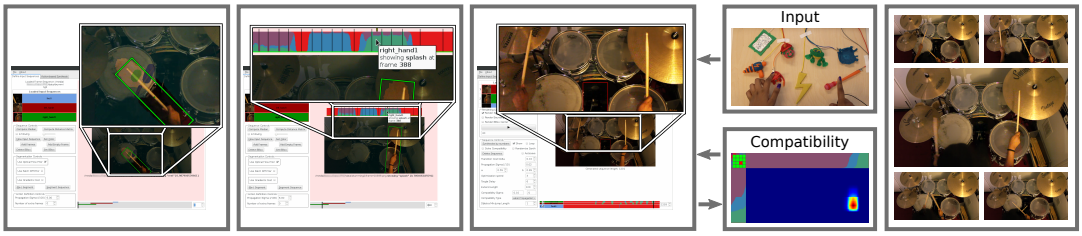
a *constrained optimization algorithm* wrapped inside our expressive new interface. Content creators can quickly improve clips that are hard to segment or loop, and check their quality live without jumping between disjoint tool-chains.

We adapt the live looping concepts to video in a prepare- and perform-structure (see Fig. 4.1). In the *preparation* stage (Sec. 4.2), we treat all moving elements as video *sprites*, *i.e.* a bendy tube of pixels in a stack of sequenced images, like Lu *et al.* [LZW*13]. We call these *actors*. If a video features only one actor, this is simply a whole-frame sprite. Each tube is then manipulated in time, while maintaining the original spatial properties, to create the output. This is done by splitting a sprite's frames into clusters of *actions* by borrowing from concepts presented in Section 3.2. Content creators and novice audiences can then choose which subset of frames to show during the second part of our approach: the live *performance* (Sec. 4.3). We show in Section 4.5 how one can request actions through a wide range of trigger interfaces, such as tangible widgets, keyboard, or paint-by-numbers, while our system ensures smooth loops within clusters and transitions between them. If necessary, content creators can also edit synthesis constraints to ensure sensible actor behavior (see Section 4.3.1).

We assume the input videos to our system adhere to the following criteria: a) the camera is stationary, b) there are no large differences in lighting over the sequence, c) the background is mostly stationary, d) the filmed *actors* are mostly well separated from each other and e) the *actions* they perform are visually distinct. We show in Section 4.6 how, despite these assumptions, our system has good *range* and can cater for widely different videos featuring various types of *actors* and *actions* and produce rich and diverse results. In Section 4.7 we demonstrate that our new medium of expression is *empowering* as it allows novice audiences to reach great results in a short time. Finally, in Section 4.8 we present informal interviews with several video-artists to determine the *balanced structure* of the live video performance.

4.1 System Overview

We design our end-to-end interface to allow content creators to quickly prototype their ideas. The more effort they are willing to invest, the higher the quality and



(a) Track and Segment **(b)** Action Definition **(c)** Responsive Synthesis **(d)** Rendered Output

Figure 4.2: Overview of our interactive video synthesis pipeline: (a) The first, *optional*, step is to track and segment the actors we wish to control, such as the two sticks and foot of the drummer in this example. (b) The user defines a set of actions for each actor by tagging example frames. Here, actions are hitting a specific drum or cymbal and resting. (c) A new video is synthesized given input commands mapped to actions and, optionally, frame compatibility information. The compatibility knowledge is learned over time, as the user tags pairs of frames, and the output is changed accordingly. (d) The synthesized sequence is composited and rendered seamlessly (using Poisson Blending [PGB03] and our custom compositing algorithm).

complexity their results can achieve. We recruited six different technical artists (game developers and visual effects artists) to help define which features in our prototype system we should focus on to best assist them in creating live video performances. Interactivity (as opposed to automation) and responsiveness were identified as crucial so we emphasize these aspects in our tool.

Broadly, videos are prepared before being used in one or more performances. With this in mind, we start by providing the necessary tools to define elements of interest which we call *actors*. These can be full-frame video sequences, such as our Toy and Candle datasets (see Fig. 4.12 and Tab. 4.1), or localized objects, such as the cars in Havana or hands in Drumming.

For the objects, we provide semi-automatic tracking and segmentation capabilities (Fig. 4.2a). We enable the user to correct any mistakes in the bounding box tracks interactively. Similarly, for separating the tracked object from the background, our tool provides previews of generated action video clips, together or in isolation. Users can then correct and influence the quality of the final segmentation by scribbling over the resulting masks.

The next step is the most critical and represents the core of our new medium

of expression. Using our simple UI (Fig. 4.2b), users associate a set of *actions* to each actor, specifying the moment in the video timeline when these are performed. For instance, each musical note in TOY, or drum hit in DRUMMING, represent semantically and visually distinct actions. Users define these by tagging a few example frames while the remaining ones are labeled automatically, based on visual similarity, using a machine learning approach. This reduces the required user input and provides almost instant feedback, allowing users to validate the automatic action association and, if necessary, refine it by tagging more examples.

A new video performance synthesizes a number of output layers, each of which corresponds to an actor. Without further guidance, our algorithm can seamlessly loop through the actor input frames by finding visually smooth transitions (similar to [SSSE00]). Users can, however, guide the live video performance by pressing keys mapped to actors' actions (Fig. 4.2c), requesting what to see and when. As we show later, this simple but powerful interaction mechanism enables more creative input methods such as *MakeyMakey* [Joy12], *synthesis-by-numbers* [HJO*01] or custom videogame logic.

Our novel and fast synthesis algorithm balances the importance of meeting users' requests with maintaining the visual quality of loop transitions, to create a new video interactively. Users can further refine the output by tagging incompatible frames or actions, so that actors interact only in desirable ways; for example, diggers should only load parked trucks (see DIGGER in Fig. 4.8). Our synthesis algorithm uses this information to improve the resulting output, completing the human-machine feedback loop that makes results possible, in response to high level triggers.

Finally, we can perform an optional post-processing step (Fig. 4.2d) to improve the quality of the output sequence recorded during the interactive phase described above. We use seamless blending to remove artifacts due to illumination changes and then merge the actor patches together with the background ensuring that the overlapping regions are handled correctly.

The following sections provide the technical and implementation details required to reproduce our system; these are followed by the results and evaluation.

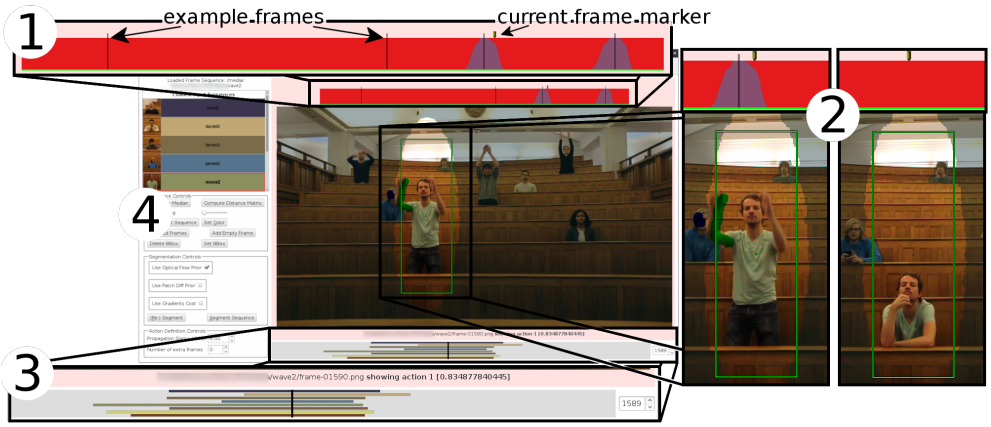


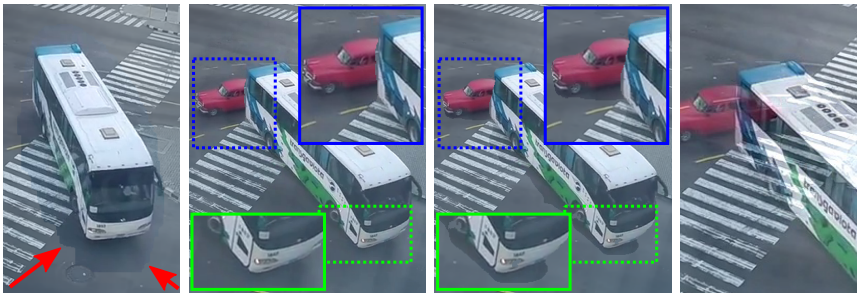
Figure 4.3: Our actor preparation user interface: (**1**) *actions* associated to each frame of a tracked object (*e.g.* the closest person is shown "sitting" in each **red** frame and "standing" in each **purple** frame, with in-between frames shown as a combination of the two colors); (**2**) two example frames with associated actions (above each and denoted by marker), bounding box and segmentation with corrective strokes (**blue** = BG, **green** = FG); (**3**) *input video timeline*: the black vertical line indicates the current frame and the colored horizontal lines indicate frames where each actor (in this case people) has been tracked; (**4**) list of tracked actors (identified by their unique color also used in (**3**)).

4.2 Actor Preparation

We now describe the steps and tools (see Figure 4.3) used to *prepare* a raw video for use during a live performance. The result of this stage is a set of *actor sequences*: video sprites associated to actions, as shown in Section 4.2.3, that can be interactively triggered during synthesis. Optionally, actors can be tracked (Sec. 4.2.1) and segmented (Sec. 4.2.2) to improve looping and increase output variability.

4.2.1 Tracking

Critical to looping algorithms is the ability to find similar frames or patches, at different points in the timeline, that can be used interchangeably to "jump" between different parts of the video. This is impossible for complex videos, such as ones with multiple, independently moving objects (see HAVANA). Methods such as [LJH13] partially address this problem by adapting their patch shape



(a) On BG **(b)** Ours final **(c)** Thresholded **(d)** Mix clone

Figure 4.4: Result of our user-in-the-loop segmentation procedure and post-process compositing: the raw image patch is placed on the original background (a) and composited using seamless cloning [PGB03] to remove lighting changes w.r.t. BG (red arrows in (a)) and our custom algorithm to resolve occlusions (b). Thresholding the BG difference introduces artifacts (c), while the "mixed seamless cloning" in [PGB03] does not resolve occlusions (d). Input © Brooks Sherman.

to best suit looping, but are prone to cutting objects, introducing visible seams. We choose to let users decide interactively which elements they may want at showtime.

In our system, objects of interest are defined by placing bounding boxes around them and tracking them over time. Tracking is a difficult problem and completely automated methods are prone to mistakes (*e.g.* drift), especially in crowded scenes like Havana. To overcome this, alongside the automatic CMT tracker [NP15], our system also provides users the tools they need to assist it when necessary (see our UI in Fig. 4.3). We chose the CMT tracker because a) it is easy and quick to correct in an interactive setting and b) it estimates both scale and orientation along with the position of the bounding box.

4.2.2 Segmentation

We then use the bounding box to constrain our custom, graphcut-based foreground (FG) segmentation algorithm. Unlike traditional approaches, we aim to composite the patches on their original background (BG). We therefore allow BG pixels to belong to the FG patch as long as *all* FG pixels are correctly classified (see Fig. 4.4a). To ensure this, users can correct any errors in the labeling by interactively scribbling over patches (the colored strokes in (2) in Fig. 4.3).

After estimating the static background as the per-pixel median of all input frames, we use the seam-finding algorithm in *Graphcut textures* [KSE*03] to separate FG from BG pixels. We use their pairwise term to conceal seams, and a novel unary term that enforces seam consistency over time and FG pixels to be within the bounding box. Formally, the unary term U for pixel s at position $\mathbf{X}(s)$ belonging to the FG in frame t is defined as

$$U\left(s,\mathbf{X}\right)=\left(1-\alpha\right)\left[-\frac{1}{2\sigma^{2}}\left\|\mathbf{X}(s)-\mathbf{X}_{\mathrm{c}}\right\|^{2}\right]+\alpha\left[1-M_{t-1}\Big(\mathcal{F}_{\leftarrow t}\big(\mathbf{X}(s)\big)\Big)\right],\tag{4.1}$$

where $\mathbf{X}_{\mathrm{c}}=(x_{\mathrm{c}},y_{\mathrm{c}})$ are the coordinates of the center of the bounding box in image space, $\mathcal{F}_{\leftarrow t}(\cdot)$ is the optical flow function that maps a pixel to its location in the previous frame [Far03] and $M_{t-1}\in\{0,1\}$ is the pixel mask (FG/BG) of the previous frame $t-1$. We use $\alpha=0.35$ against a fixed cost to the BG. User-defined scribbles fix pixels' unary cost depending on their association; see Fig. 4.4 for an example output.

4.2.3 Action Definition

The main innovation of our system is the direct mapping between arbitrary, user-defined, semantic actions and video synthesis commands. Users quickly and intuitively guide our synthesis algorithm towards their goal by issuing these commands; for instance, requesting a candle flame to flicker to the right. In contrast, traditional approaches expect users to manipulate the timelines of several clips by cutting, re-arranging and synchronizing them [JMD*12, LZW*13]; we believe this makes for far less intuitive and powerful video synthesis.

Action recognition is a well studied problem in the literature. However, existing methods focus on specific use cases, *e.g.* human actions [WRB11]. Not wanting to impose restrictions, we allow users to indicate actions of interest "by example", using our *responsive* interactive tool (Fig. 4.3). To define actions, users inspect actor sequences (potentially tracked and segmented) and indicate example frames for each action with the press of a button. Users receive *immediate* feedback on the quality of the frame-to-action association of the remaining input frames, as

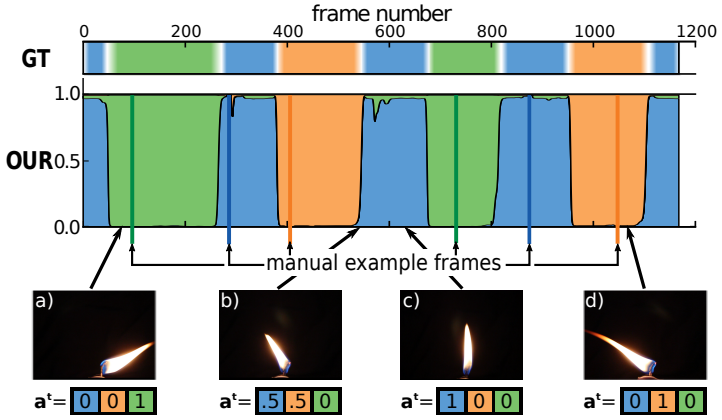


Figure 4.5: We show the automatically propagated action assignments (OUR) as opposed to the ground truth (GT). Two examples are given manually (vertical lines) for each one of the three actions (denoted with different colors). The values of the action vector $\mathbf{a}^t$ are shown for 4 example frames. Note how frame b) is correctly "softly" assigned to an action between "**left**" and "**rest**" (not present in GT).

they are automatically compared to the user-given examples.

We can view this as a fuzzy clustering problem, where each action (*e.g.* "sit" and "stand" for WAVE in Fig. 4.3) is a cluster. In practice, we represent the action visible in frame t of actor sequence $\mathcal{S}$ for which l distinct actions have been defined as an l-dimensional vector $\mathbf{a}^t$. It represents a probability distribution over the action space, so $||\mathbf{a}^t|| = 1$. Intuitively, the higher the value of the l^{th} element of $\mathbf{a}^t$, the more representative is frame t of the l^{th} action. For frames indicated as examples of a given action, $\mathbf{a}^t$ takes the form of a binary vector with a 1 for the specified action and 0's elsewhere. For instance, given the $l = 3$ actions defined for CANDLE (*i.e.* "**rest**", "**left**" and "**right**" in Fig. 4.5 and in the supplemental video[1]), a confident example frame showing the flame flickering to the left would be associated the action vector $\mathbf{a}^t = [0, 1, 0]$ (Fig. 4.5d).

We then quickly propagate the user-given information to the remaining frames using [ZGL*03]. Action vectors $\mathbf{a}^t$, with $||\mathbf{a}^t|| = 1$, are assigned to all frames, softly clustering them into different actions based on similarity to example frames.

[1] https://corneliu.co.uk/phdresults/chapter4/index.html#candleactions

The distance between each frame pair (t, t') is defined as

$$D\,(t,t') = \frac{1}{N_{\mathrm{O}}} \sum_{n=1}^{N} \Big[\mathbf{I}\big(t, \mathbf{X}(n)\big) - \mathbf{I}\big(t', \mathbf{X}(n)\big)\Big]^2 , \tag{4.2}$$

where we take the $\mathbf{L}_2$ distance between color intensities $\mathbf{I}\big(t, \mathbf{X}(n)\big)$ and $\mathbf{I}\big(t', \mathbf{X}(n)\big)$ of every pixel n. If the actor sequence has been tracked, we first place the frame's segmented patch onto the static background as shown in Fig. 4.4a. This ensures Eq. 4.2 can be used for both tracked and full frame sequences, and spatial relationships are preserved. To avoid bias due to camera-related effects, such as foreshortening, we normalize the distance measure by the number of overlapping pixels N_{O} between each frame's bounding box. We set N_{O} to the whole frame area if no bounding box is defined. For space reasons we do not discuss the propagation further. Please see [ZGL*03] and specifically their Eq.(5) for more details.

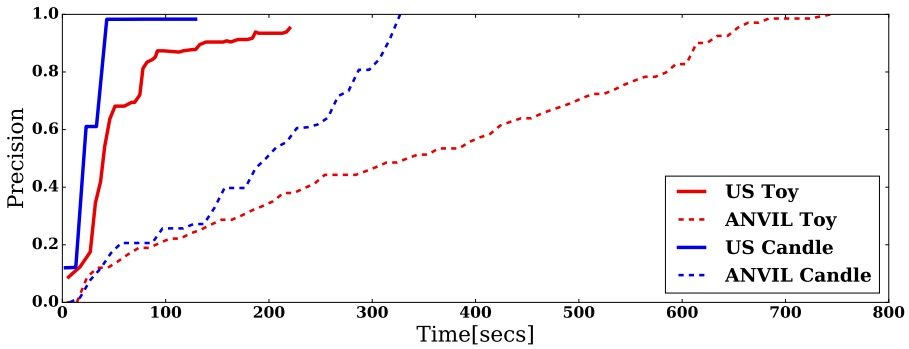


Figure 4.6: Time we spent labeling actions for datasets TOY (700 frames) with 9 different actions and CANDLE (4000 frames) with 3 different actions using our system or ANVIL [Kip14]. Precision computed w.r.t ANVIL labels.

Each input frame is associated to an action-cluster "softly" as shown in Fig. 4.5. This is critical, as frames for which no clear association exists (*e.g.* (Fig. 4.5b)), are used as in-between transitions by our synthesis algorithm. In contrast, traditional video annotation tools, such as ANVIL [Kip14], enable a similar partitioning of video sequences but with hard boundaries between *manually* defined intervals (see Fig. 4.5 GT), losing the expressiveness of fuzzy assignments in the process. Moreover, as shown in Fig. 4.6, we experienced a 2× to 3× speed-up in reaching

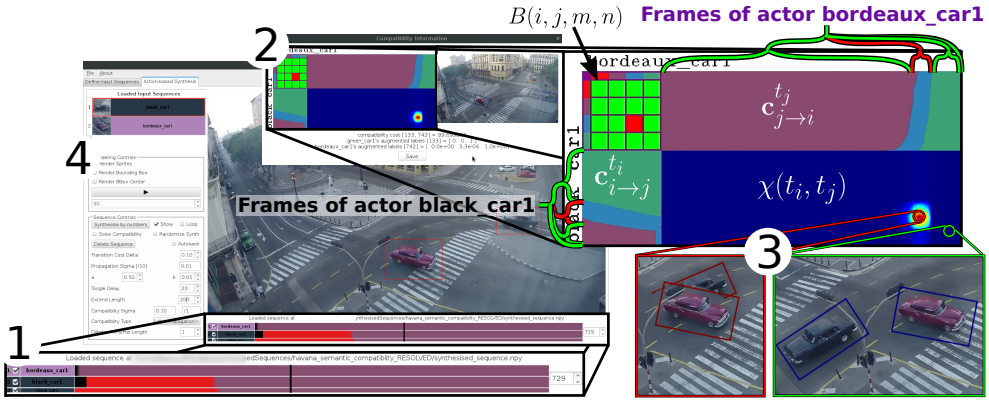


Figure 4.7: Our video synthesis user interface: (**1**) *output timeline*: lists the used actor sequences (*e.g.* 1 bordeaux car and 2 black cars) and the color coded user-given commands (*e.g.* **red** for the car to stay hidden and **purple** for it to drive through the crossing); (**2**) *frame compatibility tagging* interface: pairs of frames (previewed to the right of the compatibility info) can be tagged as compatible or incompatible; (**3**) *compatibility info* for the two selected actors (i and j): frame association to compabtibility clusters per actor ($\mathbf{c}^{t_i}_{i\to j}$ and $\mathbf{c}^{t_j}_{j\to i}$), the cluster-pair compatibility ($B(i,j,m,n)$, see Eq.4.3), for instance, here, the 3$^{\text{rd}}$ cluster of each actor (**dark cyan** above) are incompatible as denoted by **red** in the cell $(2,2)$ and the frame compatibility measure $\chi(t_i,t_j)$ from Eq 4.4 (**blue** denotes a low cost and red denotes a high cost) (**4**) list of available actor sequences (added to the output timeline shown in (**1**)) and synthesis parameters. Input © Brooks Sherman

the accuracy permitted by ANVIL (and ignoring in-between frames) thanks to the automatic label propagation from [ZGL*03].

4.3 Video Performance

In this section, we show how we synthesize a live video performance given a set of input actor sequences. During rehearsal (*i.e.* before the live performance), content creators are given the ability to interactively define the frame compatibility measure (Sec. 4.3.1), which is later used to avoid implausible outputs. Then, our optimization strategy (Sec. 4.3.2) balances user commands, frame compatibility and transition quality information to synthesize new videos in real time. Figure 4.7 shows the GUI users are presented with during the video performance stage.

4.3.1 Frame Compatibility

As we will see in Section 4.3.2, users guide the video synthesis by requesting when actors should perform actions. When multiple actors are present in the same frame however, outputs can exhibit implausible situations depending on when users issue their commands. For instance, CANDLE flames could flicker in different directions at the same time (Fig. 4.12a), a DIGGER could start loading a moving truck (Fig. 4.8a) or cars could collide in HAVANA (Fig. 4.7). In our system, these *incompatibilities* take the form of two actors' frames being composited together onto the background in the same output video frame.ravstrack

In essence, we again want to assign frames to a set of clusters fuzzily, as we did for our action definition. These clusters further decompose the actions into sub-sequences. Users mark them as (in)compatible w.r.t. the sub-sequences of other actors, indicating which sets of frames should be allowed to co-exist in the output video. Given actor sequences $\mathcal{S}_i$ and $\mathcal{S}_j$, we define the compatibility between their respective clusters m and n as

$$B(i,j,m,n) = \begin{cases} 1 & \text{if compatible} \\ 100 & \text{if incompatible} \end{cases} . \tag{4.3}$$

Initially, there is one sub-sequence cluster for each user-defined action, so m and n are in the range $[0, l)$. Later, we discuss how users marking (in)compatibilities changes the number of clusters. We initialize $B(i,j,m,n) = 1$ for all combinations of m and n. We use $\mathbf{c}_{i\to j}^{t_i}$ to denote the vector containing the probability that frame t_i of actor $\mathcal{S}_i$ belongs to clusters compatible with actor $\mathcal{S}_j$. Similarly, we define $\mathbf{c}_{j\to i}^{t_j}$ for frames of actor $\mathcal{S}_j$. We initialize $\mathbf{c}_{i\to j}^{t_i} = \mathbf{a}^{t_i}$ as it provides an initial division of the input frames, the combination of which could be incompatible. The compatibility between frame t_i of $\mathcal{S}_i$ and frame t_j of $\mathcal{S}_j$ is then defined as

$$\chi(t_i, t_j) = \sum_m \sum_n \mathbf{c}_{i\to j}^{t_i}[m]\ \mathbf{c}_{j\to i}^{t_j}[n]\ B(i,j,m,n) , \tag{4.4}$$

where $\mathbf{c}_{i\to j}^{t_i}[m]$ denotes the m^{th} element of $\mathbf{c}_{i\to j}^{t_i}$. Intuitively, the higher the probability that two frames belong to two compatible clusters, the lower the value of

(a) Without **(b)** With

Figure 4.8: Compatibility illustration. Here, the digger is requested to "load" a truck while the truck is asked to "drive" away in both cases. (a) Without frame compatibility, the two actors are free to perform these incompatible actions, with obvious artifacts. (b) With it, the digger is forced by our algorithm to "load" only when the truck actor is "parked". Input © Perfect Lazybones/Shutterstock.com

$\chi(t_i, t_j)$, denoting a low compatibility cost.

Using our GUI (Fig. 4.7) at synthesis time, users can tag pairs of frames as compatible or incompatible. Given the pair (t_i, t_j) we allow two options, which we illustrate for t_i only, as they are analogous for t_j:

1. *specialize* the compatibility by using t_i as an example for a new cluster $\tilde{m}$, re-running label propagation using the extended set of examples to compute (the now 1D larger) $\mathbf{c}_{i\to j}^{t_i}$ for all frames of $\mathcal{S}_i$, extending B by one row, setting $B(i, j, \{m \mid m \neq \tilde{m}\}, n) = 1$ and $B(i, j, \tilde{m}, n)$ according to the user input;

2. *refine* the compatibility measure by leaving B unchanged, setting t_i as a new example for the cluster $m = \underset{m}{\operatorname{argmax}}\left[\mathbf{c}_{i\to j}^{t_i}\right]$ it most likely belongs to and, again, re-running label propagation to re-compute $\mathbf{c}_{i\to j}^{t_i}$.

If the compatibility of (t_i, t_j) is changed, we assume option 1, otherwise, the user is asked to decide.

Intuitively, using the example of the two cars at the crossing in Fig. 4.7, if one chooses to *specialize*, each cluster will contain frames showing the car in different parts of the intersection. All frames showing the cars in the middle of the crossing can then be marked as incompatible and avoided in the output, making our synthesis continuously smarter.

4.3.2 Action-based Video Synthesis

An output video is composed of the frames of one or more actor sequences, re-arranged to *infinitely loop* showing specific *actions* and the *transitions* between them. We phrase our synthesis process as a labeling problem over a two-dimensional graph with D rows and K columns (see Figure 4.9). Each d row is an output layer that contains the frames $\{t_i\}$ of a specific actor sequence $\mathcal{S}_i$, re-arranged to adapt to the user's commands. Each k column represents a final output frame as the union of the frames chosen for each layer. For instance, the Toy result has one row with output frames straight from the input, while the Havana output has as many rows as controllable cars. The label assigned to each (d, k) node is the index of an actor's frame.

Users control the output video by selecting one output layer at a time (see *output timeline* in Fig. 4.7) and pressing the key associated to the action they want the actor to perform. This in turn defines for each output frame k a D-dimensional requested action vector $\{\mathbf{r}_d^k\}$, $d \in [1, D]$. We use a smooth-step function to automatically switch from the currently shown action to the one associated to the user-pressed key stroke.

Formally, our optimization strategy minimizes the energy function

$$E = \sum_{k=1}^{K} \sum_{d=1}^{D} \alpha E_{\mathrm{A}} + (1-\alpha)\Big(\beta\, E_{\mathrm{C}} + (1-\beta) E_{\mathrm{T}}\Big)\,, \tag{4.5}$$

where $E_{\mathrm{A}}, E_{\mathrm{C}}$ and E_{T} are *action*, *compatibility* and *transition costs* respectively. Intuitively, the higher the value of α, the more responsive the synthesis is to the requested actions (as E_{A} counts more towards total energy) at the expense of looping quality. This is in turn controlled by the other two terms, balanced by β. The higher its value, the more important it is to show compatible frames (E_{C}) at the expense of smooth transitions (E_{T}). Both parameters are user-tuned.

We now define the three components of our energy function. For output layer d, E_{A} is the cost of showing a frame t_i^k of actor sequence $\mathcal{S}_i$, in output frame k, based on whether the action it shows $\mathbf{a}^{t_i^k}$ matches the requested action $\mathbf{r}_d^k$ and is defined as

$$E_{\mathrm{A}}\Big(d, t_i^k\Big) = \frac{1}{2\sigma_{\mathrm{A}}^2} \Big\| \mathbf{a}^{t_i^k} - \mathbf{r}_d^k \Big\|^2 . \tag{4.6}$$

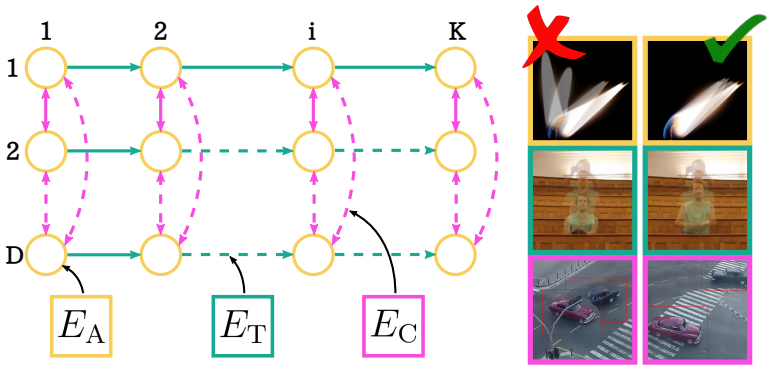


Figure 4.9: Visual representation of the energy terms in Equation 4.5 and how they relate to the synthesized video sequence. On the right, we show situations in the output video each term is designed to avoid and which to favor.

This term is designed to ensure that the candle flame in Figure 4.9 always flickers in the desired direction (*i.e.* right in the example). E_C is the cost of showing a pair of frames t_i^k and t_j^k, from output layers d and d' respectively, in the same output frame k based on the compatibility cost $\chi(\cdot,\cdot)$ from Eq. 4.4 and is defined as

$$E_C\left(d, t_i^k\right) = \sum_{j \in [1,D] \setminus d} \chi\left(t_i^k, t_j^k\right) . \tag{4.7}$$

In the example in Figure 4.9, E_C ensures the two cars do not collide with one another. Finally, E_T is the cost of showing actor $\mathcal{S}_i$'s frame t_i^k in output frame k after showing a frame $t_i^{(k-1)}$ in the previous output frame $(k-1)$ and is defined as

$$E_T\left(d, t_i^k\right) = \exp\left[\frac{1}{\sigma_T^2} D\left(t_i^{(k-1)}, t_i^k\right)\right] , \tag{4.8}$$

where $D(\cdot,\cdot)$ is defined in Equation 4.2. E_T ensures that the person sitting down in Figure 4.9 does so smoothly without visible jumps between a standing up pose and a sitting down one.

4.4 Practicalities

In this section we present a number of implementation details and practical issues that make our system successful in enabling interactive video experiences.

4.4.1 Real-time Performance

The objective in Eq. 4.5 can be solved using any discrete energy minimization solver that supports multi-label nodes and arbitrary cost functions, *e.g.* TRW-S [Kol06]. Unfortunately, TRW-S fails to converge in our experiments and our system *requires* immediate feedback to enable live performances.

Instead, we use an iterative approach similar to the block coordinate descent seen in [CK14] that minimizes our objective function locally. We define the new energy function

$$E'(d) = \sum_{k=1}^{K} E_{\mathrm{U}} + E_{\mathrm{P}}, \tag{4.9}$$

and solve it using dynamic programming, one row d at a time, to synthesize K output frames showing the actor associated to the given row. The only inter-row energy is E_{C} (Eq. 4.7), that ensures compatibility between frames of different actors, which we "bake" into the unary term E_{U}. We define it and pairwise term E_{P} as

$$\begin{aligned} E_{\mathrm{U}} &= \alpha E_{\mathrm{A}} + (1-\alpha)\beta E_{\mathrm{C}}, \\ E_{\mathrm{P}} &= (1-\alpha)(1-\beta)E_{\mathrm{T}}, \end{aligned} \tag{4.10}$$

respectively. At each iteration we minimize Eq. 4.9 for each row in the order the output layers are defined and update E_{U} to consider the already computed rows. We have found that, one iteration is enough to satisfy our three constraints and enables real-time synthesis. Note that, the result seen during a live performance might differ from the one synthesized using the full set of action requests recorded during it. This is due to the fact that, we can better optimize frame compatibilities and transitions between actions, once we exactly know what each actor will be requested to do and when. In our experiments, even long (see Table 4.1) actor sequences such as the people in WAVE or the flame in CANDLE never require more than 500ms to synthesize 400 frames. We further drastically reduce these timings (10× to 30×) by using our compression strategy described below. Typically, we synthesize less than 100 frames in $<$ 20ms every 2-3 seconds in a live performance scenario, making our system very reactive.

4.4.2 Optimization Compression

Jumps are rarely perfect, as pointed out by [SSSE00], so higher quality outputs involve fewer jumps (*i.e.* the original timeline is followed for as long as possible). With this insight, we speed up our optimization further by synthesizing a subset of the output frames using a subset of the input frames, and "filling in the blanks" using subsequent frames. For instance, we can optimize for half the output frames using only every other input frame and gain a 10× speed-up. We experimented with up to 4× compression, meaning we optimize for every 4^{th} frame, with no visible quality penalty.

4.4.3 Post-Processing Rendering

Our optional post-process first uses seamless cloning by Pérez *et al.* [PGB03], to merge each patch to the static background and remove artifacts (red arrows in Fig. 4.4a) due to illumination changes. Then, our custom compositing algorithm resolves occlusions between overlapping segmented actor patches.

As shown in Fig. 4.4a, our segmented patches often contain background pixels. This was deliberate as it allows us to retain small details such as soft shadows, and works well when patches are placed on their original background. When patches of different actors overlap in the synthesized frame (Fig. 4.4b), BG pixels may obscure FG pixels. For each pixel where this happens, we dynamically decide which patch is most likely to be FG based on its color intensity difference to the BG. This approach is more flexible and gives better results than setting a global threshold on the background difference, as shown in Fig. 4.4c. It is also more suited to our problem than the "mixed seamless cloning" in [PGB03] which does not perform well with complex backgrounds or occlusions, and introduces ghosting (Fig. 4.4d).

4.4.4 Precomputing Loops

In addition to the example output video in Figure 4.12a, we have created a controllable video-based computer game asset to use in a real game prototype.

As shown in Figure 3.1 and our results website[1], the CANDLE flame is composited within a real game level and carefully placed on top of a traditional 3D model of a candle. The game logic can also control which action the flame is showing in real time (visible in Figure 3.1b). To make this possible we have used our system to create a graph representation of the candle animation, a simplified example of which can be seen in Figure 4.10. The directed graph is composed of a number of

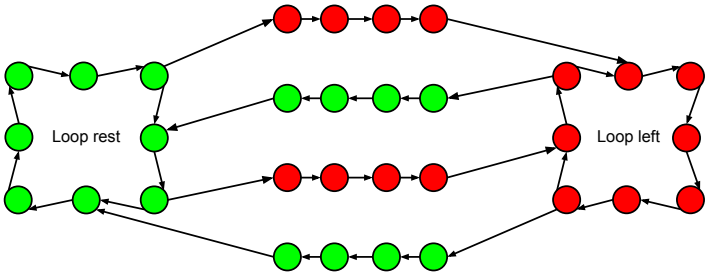


Figure 4.10: Simplified example graph used to control the candle flame video-based game asset shown in Figure 3.1

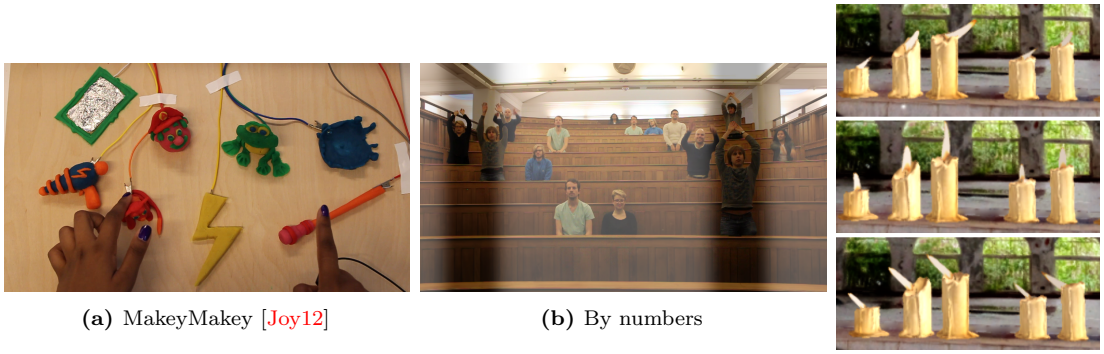
loops for each action type (green and red in the example figure) and a number of chains leading from frames in a loop to frames in the other loops. Each node in each loop is associated with the action type of the frame they are showing, while each node in the transition chains take the label of the action of the loop they lead to. The game logic can thus simply request an action type and traverse the graph to show one of the precomputed loops. When the action type changes, it can find the closest transition chain and reach a corresponding loop seamlessly.

4.5 Creative Synthesis

In contrast to existing tools, our system accepts high-level, user-defined commands that guide our video synthesis algorithm. Users need simply request when they want an actor to perform an action, enabling many alternative and fun ways of creating videos, which we now present.

Keyboard and MakeyMakey The simplest way to create a video with our system is by using a keyboard. An action for each actor can be mapped to a

[1] https://corneliu.co.uk/phdresults/chapter3/index.html#candlegame



(a) MakeyMakey [Joy12] **(b)** By numbers **(c)** Game logic

Figure 4.11: Using actions as an abstraction for synthesis commands enables new and creative ways of creating videos. Trigger commands can be given through touching a keyboard or *Play-doh* figurines (a), animated color bars (b) or context specific game logic (c).

specific key stroke which, when pressed, signals our synthesis algorithm to show frames from that category. Given the simplicity of our mapping process, we can use more creative input methods too. For instance, in Fig. 4.11a and in our supplemental video[1], an artist uses MakeyMakey [Joy12] and some *Play-doh* figurines to create a video using our DRUMMING dataset, where specific drums or cymbals are hit when the associated figurine is touched.

Synthesis by numbers Our system enables creation of video analogous to image synthesis [HJO*01]. We associate actions to solid colors, and create an animated control sequence showing those colors using any paint tool. Actions are then triggered according to the colors shown by the control sequence. For instance, Fig. 4.11b, shows an animated black bar crossing the screen from left to right. At each output frame, people in WAVE are asked to "stand" if they are under the black bar and "sit" otherwise. This allows us to quickly create a Mexican Wave. In our supplemental video[2] we show that we can easily change the control sequence to quickly synthesize completely different waves.

[1] https://corneliu.co.uk/phdresults/chapter4/index.html#drumming
[2] https://corneliu.co.uk/phdresults/chapter4/index.html#comparisons

(a) Candle **(b)** Toy **(c)** Wave **(d)** Havana

(e) Drumming **(f)** Digger **(g)** Windows **(h)** Planes

Figure 4.12: Sample output frames. Inputs to (d) © Brooks Sherman, (f) © Perfect Lazybones/Shutterstock.com, (g) © Pavel L/Shutterstock.com, (h) © Cysfilm. Results sequences shown on our supplemental results website[2].

Game Logic Our system also allows external factors to drive video synthesis. In particular, custom video-game logic can be programmed to issue commands to our synthesis algorithm based on dynamic game-related events. For instance, we have embedded a pre-computed set of outputs of our controllable Candle into a game level (see Fig. 4.11c and supplemental video[1]). Then, the game logic decides how the candle should react to its own wind simulation, by for instance, making it flicker to the left or to the right.

4.6 Results

The new medium of expression described in this chapter enables the creation of a wide variety of video performances. To stimulate the reader's creativity, we have produced a number of output videos using the system. They can be seen in our supplemental results website[2], as stills in Fig. 4.12, and are briefly described here.

[1] https://corneliu.co.uk/phdresults/chapter4/index.html#candle
[2] Visit our website http://corneliu.co.uk/phdresults/chapter4/

Dataset	*Actors*	*Actions*	*Average #frames*	*Avg Prep [s/actor]*	*Output layers*
CANDLE	1	{3}	1168	60	8
TOY	1	{9}	702	210	1
WAVE	18	{2}	1124	1320	15
HAVANA	13	{2}	587	1257	
THEME PARK	13	{1, 2}	630	820	21
DIGGER	2	{2}	160	405	2
WINDOWS	23	{2}	115	60	54
PLANES	7	{2}	105	917	10
DRUMMING	3	{2, 4, 5}	506	2440	3

Table 4.1: Example input videos. Note how some datasets contain actors with different numbers of actions. For each dataset, we show the average number of frames per actor, average number of seconds necessary to prepare them and the number of layers in the corresponding output video.

In Table 4.1, we provide information about the actors defined for each dataset and the needed user effort.

We use CANDLE as a didactic example. After segmenting the flame using pixel intensity, we define three actions ("left", "right", "rest") and thus are able to have it react according to a hypothetical breeze. Given multiple copies of a candle, as shown in our supplemental video[1], we also tag pairs of frames showing distinct actions as incompatible, such that our synthesis algorithm can ensure they all react to the breeze randomly, but in the same manner, without having to manually ensure it for each flame. Similar results are achieved from within a video-game level, as shown in Fig. 4.11c.

Range HAVANA and THEME PARK show the flexibility of our method and its ability to avoid incompatibilities. These and the subsequent examples differ from CANDLE because they cannot or would be very hard to make using existing tools. Multiple moving elements were tracked and segmented, and associated to the actions "visible" and "invisible". Thanks to our frame compatibility measure, we are able to avoid collisions between cars and people when they are visible

[1] https://corneliu.co.uk/phdresults/chapter4/index.html#candle

at the same time in the output video. Similarly, in DIGGER the user ensured a digger only loads a truck when it is parked, by tagging a few incompatible pairs of actor-frames where the truck is moving while the digger tries to load it.

With the remaining datasets, we showcase further creative interactions with our system. Using WAVE, we create a Mexican Wave simply by creating a control animation as seen in Fig. 4.11b. We can then quickly alter the result by simply changing the control sequence, to add a second subsequent wave, one in the opposite direction, or even an interlaced one. In DRUMMING, we can control a drummer playing his instrument by simply touching *play doh* figurines representing funny sounds, while with TOY we can create a video showing specific songs being played onto a colorful xylophone after filming random notes being hit. WINDOWS allows us to map windows on a building facade to pixels in a grid, and render a compelling game of Tetris by manipulating the light switches. With PLANES, airplanes take off in sync with a user hitting the spacebar to the rhythm of a well known videogame theme-song. Finally, we used HAVANA to create the CounterLoop videogame (see Section 4.6.1). All these outputs and use-cases are prepared with the same workflow, qualitatively demonstrating its range.

4.6.1 Counter Loop

As described throughout this thesis, one overarching goal is to make the experience of watching a video more game-like. We took this idea to the extreme and built an actual video game called Counter Loop as shown in Figure 4.13 and on our supplemental video[1]. In this game, the goal of the player is to control a number of cars and help them safely cross an intersection. At each level (shown in the top left corner in Figure 4.13), the player controls the car marked by the golden arrow. They can decide when to let the car drive across the screen and when to stop by pressing the space bar on the keyboard. Once a car has crossed the intersection, the next car appears and can be controlled while the game plays back the recorded animation of the previously controlled ones. The higher the level, the larger the number of cars on screen and thus the harder it is to cross the intersection. The user receives points based on how fast she can control the cars to

[1] https://corneliu.co.uk/phdresults/chapter4/index.html#counterloop

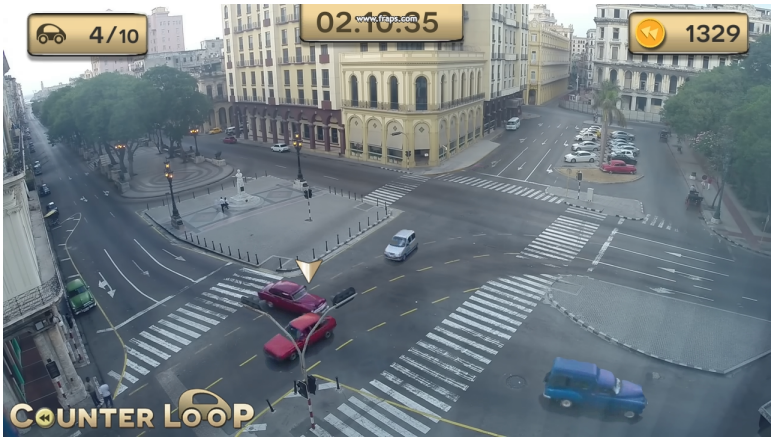


Figure 4.13: Counter Loop: a videogame using video sprite assets made using the system described in this chapter. The player must traverse the crossing using the marked car as quickly as possible without crashing against the other cars.

safety through the intersection, so there is a push to rarely stop the controlled car to let others pass. While the logic and the UI have been created using a traditional game engine, the controllable cars have been created using our system, showing how video-based game assets can be used in traditional computer games. They have been tracked and segmented using the procedures described in Sections 4.2.1 and 4.2.2 and their speed has been normalized as described in Section 3.2.2.

4.7 Empowerment Evaluation

In this section, we aim to evaluate whether our new medium of expression is *empowering*, *i.e.* whether end-users (both content creators that processed the input video and novice audiences) can quickly and easily express their creativity by creating novel videos. We have chosen to only focus on the live performance part of our system as that is the focus of our medium of expression and where creativity comes most into play. Initially, we looked at our our closest competitors [LJH13, LZW*13, JMD*12] and attempted to recreate our Mexican Wave output (Fig. 4.12c). Liao *et al.* [LJH13] can deal with complicated scenes but it merely finds the best possible looping patches. It does not allow the user to choose where to loop or when people should sit and when to stand. The system of Lu *et al.* [LZW*13] can allow us to splice together different sub-clips and re-arrange

them. However, they create unnatural speed-ups or slow-downs when people need to sit for longer or less than the input video, because of their time scaling algorithm, and do not account for transitions between the clips as our looping does automatically.

We informally compare our system to *Cliplets* [JMD*12] by recreating an 8-actor Mexican Wave (see supplemental video[1]) and one candle flame using CANDLE. Similar to [LZW*13], *Cliplets* works by defining, manipulating, and arranging layers of video clips to create the output. Each layer shows one looping animation (*e.g.* an actor performing one action) or input frames as captured (which we used to transition between actions of the same actor). For instance, a video of a flame flickering left and then right, requires three layers: a) "loop flame left", b) "playback flame going from left to right", c) "loop flame right". Users must manually define when to show each loop and its length and find transition frames in b) such that there is no visible jump when hiding layer a) to show b) and when hiding b) to show c). This very time consuming process is slowed further if the result needs changes, as changing looping time or animation order requires carefully re-arranging layers and redefining transitions, effectively starting over. In contrast, our system enables live performances after a one-off preparation stage. The output is created as an endless stream and the user is free to play-act and improvise in real time, an invaluable ability unique to our system. Using *Cliplets*, it took us 4× and 9× longer to recreate WAVE and CANDLE respectively. This is mainly due to manually inspecting the video to find the right subset of frames to loop through or use as transitions between animations.

For reasons discussed above, several methods [JMD*12, LJH13, LZW*13] were not able to successfully recreate our outputs. Separately from range, we want to assess whether our action-based synthesis *empowers* users' creativity [Ols12] and helps express it better and faster than baselines. We therefore compare against Adobe After Effects (AE) because, with proper training, it gives users commercially-accepted tools that should reproduce our results. We gathered 6 novice users, that had never used either system, and asked them to recreate the Mexican Wave sequence. In particular, they were instructed to create a left-to-right wave, some idle animation in the middle (*i.e.* sitting people) and a final

[1] https://corneliu.co.uk/phdresults/chapter4/index.html#comparisons

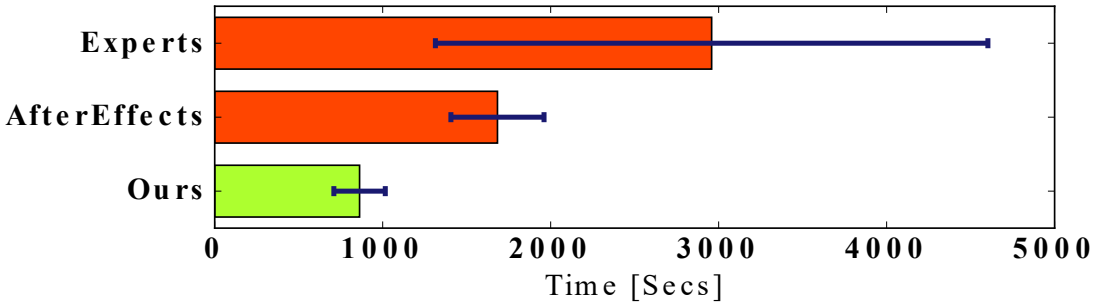


Figure 4.14: Average timings necessary to replicate a variation of our Mexican Wave result. From top to bottom: *expert* users of NukeStudio, Blender, and AfterEffects; novice users of *AfterEffects*; novice users of *our* system.

right-to-left wave. After relevant training, half of the users used our system while the remaining half performed the same task using AE. Both sets of users were given the same 7 sprites as input that had already been tracked and segmented.

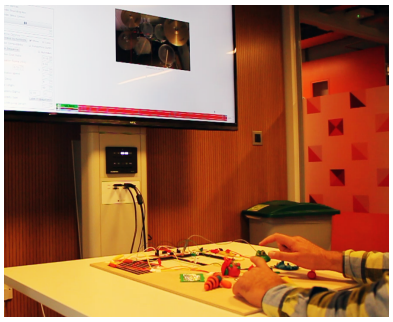
Figure 4.14 shows that users of our system were roughly twice as fast, indicating that our system is indeed easy to use. In their words, they really enjoyed the simplicity with which actions are defined, the responsive visualization (Fig. 4.3), and the immediate video feedback that comes with action requests during synthesis. In the supplemental videos[1], we qualitatively show that the AE results are inferior to ours. This is because our system automatically finds the best transitions between actions, while the AE users need to manually align the clips showing the sitting and standing actions and decide when to transition between them. Finally, we also asked three expert Nuke Studio, Blender, and AE users to recreate the same sequence. Their timings (also in Fig. 4.14) show a larger variability depending on their willingness to find optimal-looking jumps. In fact, they were given the same inputs and task as the novice AE users, but no further instructions, and their results are of varying quality.

4.8 Discussions with Artists

To assess the *balanced structure* [Ols12] of our system, we informally interviewed three digital artists, mostly involved with game development or live

[1] https://corneliu.co.uk/phdresults/chapter4/index.html

performance design, and introduced them to our system (see inset image using our DRUMMING demo). They agreed that our system takes a large step toward making "video composition more like playing a musical instrument", enabling live performances with immediate video feedback, such as seen in DRUMMING. We were surprised that one asked to sacrifice video quality for better responsiveness, especially if sound feedback is present. As shown in our supplemental video[1], we were able to cater to this request by favoring E_A(Eq. 4.5) at the expense of good looking transitions. The result can then be improved, immediately after recording the user commands, by re-synthesizing the sequence with the default video settings.



Interestingly, artists saw our system as a "sketching tool" for quick prototyping, such as seen using *synthesis by numbers* to create the variations of the Mexican WAVE. In fact, experimenting with choreographies was a suggested use case, such as filming dancers improvising and re-arranging their moves using our system after the fact. They also expressed the desire to have the synthesis algorithm as part of game engines, as they feel it gives them important control over sprite synthesis. When shown our CANDLE video, they immediately recognized its value for game development and suggested further content, such as water drops into puddles. Finally, they suggested a number of "shared experiences" for teaching and training that our system would make possible. For example, a trainer could decide which exercises people should perform and give them live video tutorials, or could trigger traffic scenarios.

4.9 Conclusions

In this chapter, we have presented a system that facilitates a new medium of expression, where videos are created much like live audio looping is composed. Users define actors, optionally tracking and segmenting them, and associate actions to triggers, which can take the form of multiple interfaces. Users can also flag

[1] https://corneliu.co.uk/phdresults/chapter4/index.html#instant

combinations of actions they do not want actors to perform together and explicitly handle them as part of our synthesis optimization. Our work-flow helps both novices and advanced users to prepare their footage and, for the first time, turn it into interactive live video performances.

We have successfully tackled some of the issues described in Chapter 3. At the core of our system is the ability to show the input frames in any possible order and therefore loop (Section 3.1) a video indefinitely. As discussed in Section 3.2 however, looping is not interesting enough so we have added the ability to interactively control the process of synthesizing a new sequence by means of semantically meaningful actions. Our synthesis algorithm can run in real time (Section 3.3) as demonstrated in our results section. In order to widen the range of videos we can process and control, we provide users the ability to track and segment individual filmed objects (as discussed in Section 3.4). Finally, all these elements come together as an end-to-end interactive authoring system, as described the necessity of in Section 3.5.

Limitations The quality of our results, ultimately depends on the input videos and the users' willingness to invest the necessary effort to process them. The longer the actor sequences, the greater the variability and coverage of situations. For instance, there are no clean transitions between hitting some notes in Toy and the rest position, because the mallet hand rarely leaves the view in the input video, resulting in occasional jumpy animation. In general, this holds for short videos where there is too much variability but not enough coverage. This problem could be tackled with interpolation and morphing techniques similar in spirit to [SLWSS15]. Additionally, there is always a trade-off between how quickly the synthesis shows the desired action, and how smooth the transition looks. Being able to successfully camouflage bad jumps would reduce the time necessary to transition between actions. It would also remove the input lag between the button press and the on-screen response, critical for live performances such as Drumming.

Our system allows users to track and segment objects, a process which becomes tedious and time consuming when the background changes or there are occlusions. Although we don't require these features and they could be performed outside of our system, we aim at finding novel solutions to speeding them up and improve

robustness. Moreover, the techniques we develop in Chapter 5 could trivially be integrated with our system to enable automatic tracking and segmentation. Finally, this system can effectively only synthesize new 2-dimensional frames, but as we discussed in Section 3.6, we would eventually like to introduce 3-dimensional control of the created assets. Again, the solutions we devise in Chapter 5 aim to tackle this limitation.

Chapter 5

Multi-view from single-view

In Chapter 4 we have described an end-to-end system that gives us the ability to turn a static-camera video into a set of video assets that can be controlled in real time by simply pressing buttons on a keyboard. It takes ideas from Chapter 3 and improves upon them to enable compelling and new interactive video experiences. It however has two important limitations which we would like to address in this chapter. First, our tool was designed to encourage content creators to help our automatic algorithms where necessary. Particularly during tracking and segmentation, the amount of needed effort becomes occasionally prohibitive so in this chapter we are interested in ways to avoid this issue. Second, we would like to create video game-like experiences from traditional videos, however our system is currently limited to a two-dimensional world as it can only manipulate the pixels of 2D video frames. Could we infer information about the 3D world from a sequence of images and if yes what new effects and interactions would it enable?

In our Counter Loop game prototype described in Section 4.6.1, players are given the ability to decide when cars passing through a road crossing should move and when they should stop. As we can only reshuffle the frames of the given video, the user input can only influence when we play back the images where the cars have been tracked and segmented from the static background. In modern video games, players can interact with on-screen objects in much more compelling ways by, for instance, changing the path a car follows or moving the virtual camera to see the scene from a different angle. Moreover, despite our best

efforts, there is still a considerable amount of effort involved in processing and preparing each car before they can be used as assets in Counter Loop. Users need to interactively track (Sec. 4.2.1) and segment (Sec. 4.2.2) each object separately and then manually define camera properties to normalize their speed and define their footprint (Sec. 3.2.2).

We dedicate this chapter to presenting our efforts towards mitigating these limitations by both reducing necessary user effort while maintaining high quality and enabling new types of interaction in video experiences by inferring knowledge about the 3D world from sequences of 2D images. In Section 5.1 we describe a completely automatic method for estimating the camera position and orientation with respect to a planar surface placed in the 3D world. We then show in Section 5 how to extend a traditional image-space 2D bounding box multi-object tracker with 3D information. We can automatically approximate the size of objects using three-dimensional bounding volumes and track their position and orientation as they move on the ground plane. In Section 5.3 we discuss two prototype applications that are enabled by having tracked objects in 3D as opposed to the traditional 2D image-space tracks. We end with Section 5.4 where we present our results and discuss them in Section 5.5.

5.1 Camera Estimation

Given a sequence of images captured over time from a static camera, we want to reason about the behavior of moving objects such as the cars in HAVANA. In order to do this, we need to recover the transformation that points in the 3D world are subjected to when they are projected to points on the 2D image plane. More formally, we define the function $P(\mathbf{x}')$ which projects 3D world-space points $\mathbf{x}'$ into 2D image-space points $\mathbf{x}$ according to the following transformation:

$$P(\mathbf{x}') = \mathbf{K}\mathbf{P}\tilde{\mathbf{x}}' = \tilde{\mathbf{x}}, \tag{5.1}$$

where we use the tilde notation, such as in $\tilde{\mathbf{x}}$, to represent homogeneous coordinates of corresponding vectors. The intrinsics matrix $\mathbf{K}$ represents the internal camera parameters such as focal length and principal point. We do not attempt to

estimate $\mathbf{K}$, a process known as camera calibration, and instead use the following approximation introduced in [PVGV*04]:

$$\mathbf{K} = \begin{bmatrix} w+h & 0 & w/2 \\ 0 & w+h & h/2 \\ 0 & 0 & 1 \end{bmatrix}, \tag{5.2}$$

where w and h are the width and height of the video frames respectively. In the absence of strong zoom, wide-angle lenses and random frame cropping, Pollefeys *et al*. [PVGV*04] show that images are quite well approximated by this formula. We adopt their solution as it avoids an extra step in the camera estimation process. This would be especially complicated as we often do not have access to the camera used to film the footage such as HAVANA which was found on the Internet.

The 3×4 matrix $\mathbf{P} = \begin{bmatrix} \boldsymbol{\Omega} & \boldsymbol{\tau} \end{bmatrix}$ represents the camera pose in terms of its position $\boldsymbol{\tau}$ and orientation $\boldsymbol{\Omega}$ with respect to the world coordinate system. The most common way to recover $\mathbf{P}$ is through a process called *Structure from Motion* (SfM) (*e.g.* [SSS08, MMMO, SF16]) which works by ingesting a set of images of a static scene captured from multiple viewpoints. It then finds points visible in multiple images and jointly triangulates their 3D positions and estimates the camera pose that best explains the 2D location those points project to in the image plane.

In our case, the camera is static, which means we can rely on a number of alternatives to SfM. Point-based 3D modeling, such as the one we describe in Section 3.2.2, estimate the camera pose w.r.t. some geometry proxy for the captured scene (*e.g.* the ground plane). Methods such as [CRZ00] on the other hand, find the camera pose based on the geometric properties of parallel lines projected onto the image plane. Relying on the assumption of a Manhattan world, they intersect image-space lines that are parallel in the real world to find *vanishing points*. As seen in [Tar09], they can be seen as *vanishing directions* and used to recover the camera pose. Typically, vanishing directions are found by finding prominent edges in image space (*e.g.* [Can86]), fitting line segments to them and grouping them based on whether they are parallel in the real world to find orthogonal sets of lines. As shown in Figure 5.1, even for man-made

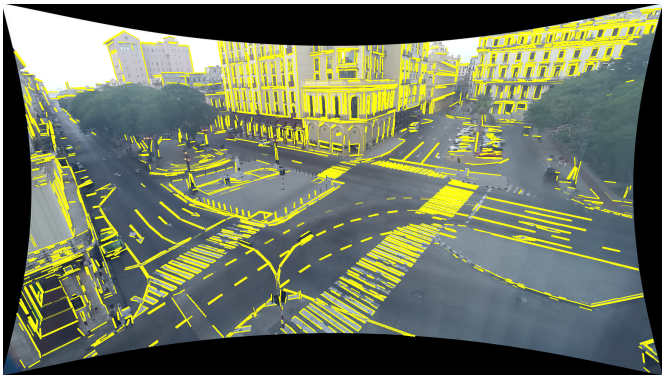


Figure 5.1: Line segments on the HAVANA static background found by [vGJMR12]. Note how even for a human it is hard to find orthogonal pairs of parallel lines.

environments such as seen in HAVANA, finding orthogonal vanishing directions can be problematic due to noisy line segment proposals and broken Manhattan-world assumption.

As pointed out in [WZJ16], making the crucial assumption that orthogonal vanishing directions can be inferred from an image, has contributed to stagnating the development of horizon line estimation methods. Moreover, the horizon line is defined as the union of *all* vanishing points defined by parallel lines placed on the ground plane, so finding orthogonal ones should not be a requirement. In fact, methods such as [Tar09, ZWJ16] are relatively successful at finding the horizon line even though they are not guaranteed to find orthogonal vanishing directions. Moreover, recent methods based on deep learning such as [ZWJ16] do not rely on vanishing direction at all. Based on this observation, we propose an automatic method for recovering the camera pose with respect to a world-space plane given only the image-space horizon line and without the need for orthogonal vanishing directions.

Let us define the world-space ground plane $G = (\mathbf{p}', \boldsymbol{\omega})$ defined in terms of its three-dimensional origin point $\mathbf{p}'$ and orientation $\boldsymbol{\omega} = \begin{bmatrix} \alpha & \beta & \gamma \end{bmatrix}$. We set $\mathbf{p}' = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$ and its image-plane projection $P(\mathbf{p}') = \mathbf{p} = \begin{bmatrix} w/2 & h/2 \end{bmatrix}$. We can

define the camera pose w.r.t. the plane by using

$$\boldsymbol{\Omega} = \begin{bmatrix} -\cos\beta\cos\gamma & \cos\beta\sin\gamma & -\sin\beta \\ \cos\alpha\sin\gamma + \sin\alpha\sin\beta\cos\gamma & \cos\alpha\cos\gamma - \sin\alpha\sin\beta\sin\gamma & -\sin\alpha\cos\beta \\ \sin\alpha\sin\gamma - \cos\alpha\sin\beta\cos\gamma & \sin\alpha\cos\gamma + \cos\alpha\sin\beta\sin\gamma & \cos\alpha\cos\beta \end{bmatrix} \tag{5.3}$$

and

$$\boldsymbol{\tau} = \begin{bmatrix} 0 \\ 0 \\ -\lambda \end{bmatrix} \tag{5.4}$$

where $\lambda = 1$ is a scaling factor which can be manually set to give a metric-scale camera pose. While we do not need this for our use cases in the next sections, we allow users to set λ in our tool described in Section 5.1.1. Finally, we define two orthogonal pairs of parallel lines on the world-space ground plane (*e.g.* see **red** and **yellow** lines in Figure 5.2) and project them into the image space using $\mathbf{K}$ as defined in Eq. 5.2 and the current estimate of $\mathbf{P}$ defined in terms of $\boldsymbol{\Omega}$ and $\boldsymbol{\tau}$ from Equations 5.3 and 5.4 respectively. The two orthogonal vanishing points $\mathbf{v}_X$ and $\mathbf{v}_Y$ are found by intersecting the image-space parallel lines (see Figure 5.2 for a visual example).

In order to estimate $\boldsymbol{\omega}$, we devised an algorithm that matches the *estimated* horizon line defined by $\mathbf{v}_X$ and $\mathbf{v}_Y$ to a *target* horizon line estimated by an off-the-shelf method. We have chosen [Tar09] as it works well in our examples and code is readily available however, any, more accurate, alternative would do (*e.g.* [ZWJ16]). We use the Nelder-Mead simplex search algorithm [NM65] to minimize the energy function

$$E_H(\boldsymbol{\omega}) = \lambda_H \left| \frac{\mathbf{v}_Y - \mathbf{v}_X}{||\mathbf{v}_Y - \mathbf{v}_X||} \times \frac{\breve{\mathbf{v}}_Y - \breve{\mathbf{v}}_X}{||\breve{\mathbf{v}}_Y - \breve{\mathbf{v}}_X||} \right| + (1 - \lambda_H)||\breve{\mathbf{v}} - \breve{\mathbf{v}}_\perp(\mathbf{v}_X, \mathbf{v}_Y)|| \tag{5.5}$$

where $\breve{\mathbf{v}}_Y$ and $\breve{\mathbf{v}}_X$ are two points on the target horizon line and $\breve{\mathbf{v}}_\perp(\mathbf{v}_X, \mathbf{v}_Y)$ is the perpendicular projection of an arbitrary point $\breve{\mathbf{v}}$ on the target horizon line onto the estimated horizon line defined by $\mathbf{v}_X$ and $\mathbf{v}_Y$. Figure 5.2 shows a visual representations of the terms in Equation 5.5. Intuitively, the left hand-side cross product in $E_H(\boldsymbol{\omega})$ represents the angular distance between the estimated and

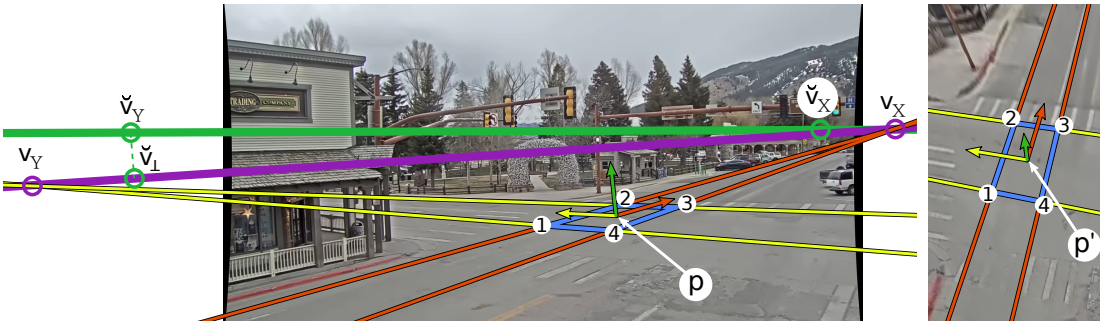


Figure 5.2: Camera pose estimation based on horizon line and ground plane. The two orthogonal pairs (marked in **red** and **yellow**) of parallel lines, can be used to find the two vanishing points $\mathbf{v}_X$ and $\mathbf{v}_Y$ (marked as a **purple** circles) which in turn define the **green** horizon line. The rectangle defined by the four numbered intersection points between the colored lines, can be used to define a perspective transformation between the image and ground planes. On the right, we show a birds-eye view of the image projected onto the world space ground plane as evidenced by the arrows marking the three-dimensional axes. The origin of the 3D coordinate system is denoted by $\mathbf{p}'$ and its image-space projection by $\mathbf{p}$.

target horizon lines while the right hand-side forces the two lines to be close to one another. We set $\lambda_H = 0.995$ for all our experiments. Note that, for visualization purposes, we flip the Z axis if the result of the optimization has it pointing downwards in image space.

5.1.1 User-in-the-loop estimation

The automatic method described above is agnostic to scale, which means that a unit in world space is completely arbitrary and does not correspond to metric units. While it is not an issue for the use cases described in the following sections, we briefly describe here the tool we developed for metric camera estimation and point-based scene modeling.

Figures 5.3 and 5.2 give a brief overview of our tool. Users can define two orthogonal pairs of parallel lines (shown in **red** and **yellow** in Figure 5.2) on the ground plane by dragging handle points (not shown). We intersect each pair of parallel lines with one another to find two orthogonal vanishing points $\mathbf{v}_X$ and $\mathbf{v}_Y$ (**purple** circles in Fig. 5.2). We found the **green** horizon line in Figure 5.2 which connects them to be invaluable for visually gauging the correctness of the estimated

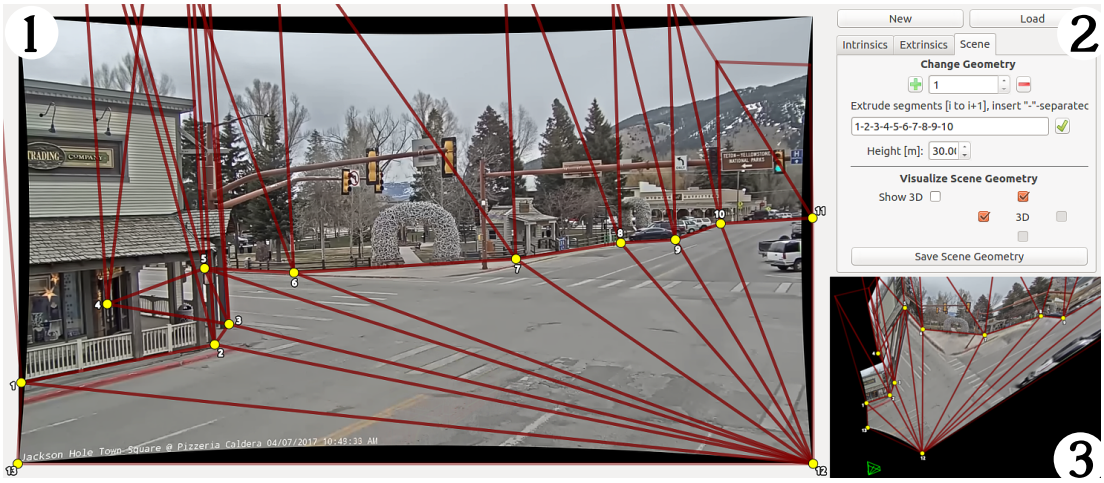


Figure 5.3: Tool for manually estimating camera pose and scene geometry. (**1**) *work area*: users can place the yellow points to define corners in the triangular mesh (shown as **red** lines) that represents the scene geometry; (**2**) *control panel*: manually defined parameters specific to one of three stages: *Intrinsics* stage specifies camera intrinsics $\mathbf{K}$ and lens distortion parameters; *Extrinsics* stage defines parallel lines on the ground plane used to estimate the camera pose $\mathbf{P}$ and is shown in Figure 5.2; *Scene* stage is shown here and it defines the scene mesh as shown in (**1**); (**3**) *3D viewer*: preview of the scene geometry from a birds-eye perspective. The capture camera viewpoint is shown as a **green** frustum.

camera pose. Given the four numbered corners of the **blue** rectangle in Figure 5.2 resulting from intersecting the four user-defined lines, we estimate the projective transformation $\mathbf{P}$ using the *perspective-three-point* solution of [GHTC03]. As an alternative, users can manually define a target horizon line (as seen in Figure 5.2) and we can estimate the camera pose automatically using the method described in Section 5.1. The tool gives users the option to define the metric scale of the rectangle for a more meaningful calibration. Given $\mathbf{K}$ and $\mathbf{P}$, we can project the image-space points $\mathbf{x}$ onto the world-space ground plane

$$\tilde{\mathbf{x}}' = (\mathbf{K}\begin{bmatrix}\boldsymbol{\Omega}_0 & \boldsymbol{\Omega}_1 & \boldsymbol{\tau}\end{bmatrix})^{-1}\tilde{\mathbf{x}} \tag{5.6}$$

where $\tilde{\mathbf{x}}'$ is a two-dimensional point on the 3D ground plane represented in homogeneous coordinates and we use $\boldsymbol{\Omega}_i$ to denote columns of the rotation matrix $\boldsymbol{\Omega}$. This allows us to project the full image onto the ground plane to view the

scene in 3D. Moreover, we can quickly define a triangulated mesh representing the scene by clicking on image-space corners (yellow points in Fig. 5.3) to form a planar polygon on the ground plane and extruding its edges along the Z axis to define the triangulated scene mesh shown in **red** in Figure 5.3. Please see our supplemental video[1] for a demo.

5.2 Well-grounded Tracking

Object tracking is an actively researched area of computer vision which aims to find and track moving objects filmed over time. Specifically, trackers are tasked with determining the *pose* of one or more objects in each frame of an input video. The pose is most commonly an axis-aligned bounding box or in some cases an oriented one, but it can vary from a single pixel location representing the center of the object to a per-pixel segmentation mask. Despite many recent improvements in recent years, visual object tracking is still a very challenging problems for a variety of reasons. Most trackers rely on finding discriminative visual features, such as texture, to detect and track objects. When they move across a scene, objects can deform, be seen from different angles or get occluded (partially or completely), which drastically changes their appearance and makes tracking unreliable.

One of the biggest reasons for appearance changes is the fact that video frames are a heavily distorted and constrained view of the real world. The appearance of a filmed object is view-dependent, and changes when the object changes position and orientation with respect to the camera. If we could eliminate at least some of the effects due to perspective projection, we would eliminate a number of issues trackers are faced with. For instance, if we had a top-down view of a scene, we would not have to worry about occlusions anymore (assuming all objects move on the same plane). Moreover, the movement and scale of objects would not be view-dependent anymore so a constant real-world change would correspond to a constant change in our observations, which is not true for perspective cameras due to foreshortening. As we show in Section 5.1, we can now model the perspective transformation performed by a camera, so can we leverage this new information to extend image-space tracking?

[1] https://corneliu.co.uk/phdresults/chapter5/index.html#camerapose

5.2.1 Multiple objects 2D Tracks

As previously mentioned, in this chapter we are interested in finding new ways to infer more information about what objects look like and how they move in order to enable new interactive video experiences. We choose to not focus on designing a new 2D tracker and instead aim to use camera pose information to extend and improve upon existing ones. As we discuss in Sections 5.3 and 5.4, the output of our new algorithm described in Section 5.2.2 enables a number of new applications by smoothing tracking results over time and inferring the shape of the tracked objects.

In choosing a tracker to use as an input, we focused on three criteria: fast, readily available and completely automatic. Traditionally, single object trackers require a first manual step where users mark the object they want to track in the first frame. Instead we look at multi-object trackers which by definition aim to first detect objects they need to track. By these criteria, we have found [BES17] to be a perfect fit to our needs. It has been proven successful at both [WDC*15, MLTR*16] tracking challenges and it is extremely fast as it does not use any image information and simply relies on the *Intersection over Union* (IoU) score between boxes found in subsequent frames. Crucially, unlike some of its competitors, the IoU Tracker [BES17] is completely agnostic to object type as it takes per-frame bounding boxes from existing object detectors and rely on the assumption that they are highly reliable and robust, which is ever more true thanks to recent advances in Deep Learning. The object detector we have chosen is the very fast state-of-the-art *Evolving Boxes* (EB) [WLW*17] car detector. Crucially, in the next sections, we do not make any assumption on the type of objects we are tracking so our choices could be easily swapped out for something else. We show the input tracks for all our datasets from Table 5.1 on our website[1].

5.2.2 Estimating and Tracking Cuboids on the Ground

In this section, we describe an algorithm for estimating an object's *shape*, defined as a three-dimensional bounding volume (a.k.a. cuboid) and its *pose* over time. Here, we parametrize the shape $\mathbf{s} = \begin{bmatrix} s_L & s_W & s_H \end{bmatrix}$ as a 3D vector of sizes (length,

[1] https://corneliu.co.uk/phdresults/chapter5/index.html

width and height respectively) as shown in Figure 5.5a. The pose at time t is denoted by $\mathbf{p}_t = \begin{bmatrix}\mathbf{x}'_t & \theta_t\end{bmatrix}$, where $\mathbf{x}'_t = \begin{bmatrix}x'_t & y'_t & 0\end{bmatrix}$ is the 3D position and θ_t the orientation of the cuboid as it travels on the ground plane. The orientation θ_t is defined as a rotation angle about the Z axis.

Initialization

The input to our algorithm are a set of N image-space 2D bounding boxes such as the ones described in Section 5.2.1 and the result of camera calibration from Section 5.1. Note that, as we mention in Table 5.1, we first discard frames where the 2D bounding boxes are too close to the image frame as they are likely to contain objects that are partially outside of the video frame. We initialize the trajectory $\mathbf{T} = \begin{bmatrix}\mathbf{x}_t, t \in \begin{bmatrix}1, N\end{bmatrix}\end{bmatrix}$ where the set of image-space points $\mathbf{x}_t$ are defined by taking the middle point of the bottom-most edge of the tracked box at each time t. Using Equation 5.6, we recover the 3D-world trajectory on the ground plane $\mathbf{T}' = \begin{bmatrix}\mathbf{x}'_t, t \in \begin{bmatrix}1, N\end{bmatrix}\end{bmatrix}$. Before computing the initial value for all orientations θ_t, we fit a two-dimensional (all values of the z-coordinate are set to 0) spline to the trajectory $\mathbf{T}'$ to yield a smooth trajectory $\hat{\mathbf{T}}' = \begin{bmatrix}\hat{\mathbf{x}}'_t, t \in \begin{bmatrix}1, N\end{bmatrix}\end{bmatrix}$. We then sample the spline at $t + \epsilon$ to yield $\hat{\mathbf{x}}'_{t+\epsilon} = \begin{bmatrix}\hat{x}'_{t+\epsilon} & \hat{y}'_{t+\epsilon}\end{bmatrix}$. The orientation angle at each point on the smooth trajectory is defined as $\hat{\theta}_t = \arctan(-\hat{y}'_d/\hat{x}'_d)$ where $\hat{\mathbf{x}}'_d = \begin{bmatrix}\hat{x}'_d & \hat{y}'_d\end{bmatrix} = \begin{bmatrix}\frac{\hat{x}'_{t+\epsilon} - \hat{x}'_t}{\eta} & \frac{\hat{y}'_{t+\epsilon} - \hat{y}'_t}{\eta}\end{bmatrix}$ is the direction in which the object has moved and η normalizes $\hat{\mathbf{x}}'_d$ to unit length. The pose at each time step t is initialized using the positions and orientations derived directly from the smooth trajectory so we set $\mathbf{p}_t = \begin{bmatrix}\hat{\mathbf{x}}'_t & \hat{\theta}_t\end{bmatrix}$. Finally, we initialize the cuboid shape $\mathbf{s}$ to the shape that best fits the input tracked 2D boxes while still maintaining car-like proportions. In other words, we use the bounding volume size that minimizes the two energy functions in Equations 5.11 and 5.12.

Spline-based smoothing

As mentioned above, in order to smooth out the potentially very noisy sequence of points that defines an object's trajectory $\mathbf{T}'$, we fit a two-dimensional spline which, when regularly sampled, yields a new set of points $\widehat{\mathbf{T}}'$. In particular, we fit the spline to the subset of points $\mathbf{x}'$ in $\mathbf{T}'$ where the curvature is below 10°. The reasoning behind this choice is that objects do not change their moving direction

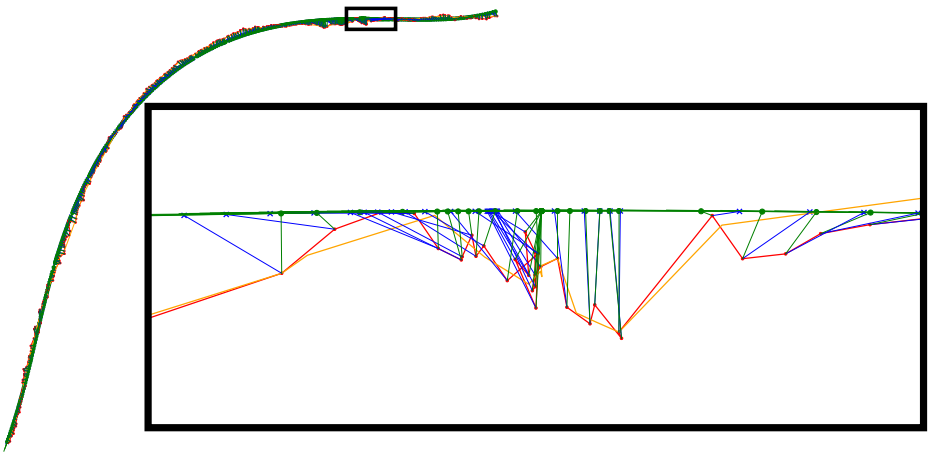


Figure 5.4: Top down view of smoothing a noisy trajectory using splines. The input trajectory $\mathbf{T}'$ is shown in **red** while the simplified curve by thresholding based on curvature is shown in yellow. In **blue** we show the mapping between the input **red** points and the smooth trajectory $\widehat{\mathbf{T}}'$ while we show in **green** our final smooth trajectory $\check{\mathbf{T}}'$

drastically between subsequent frames so a steep curvature can only be due to noise in the input trajectory. It is important to note that, even if the spline correctly approximates $\widehat{\mathbf{T}}'$, it does not guarantee points $\mathbf{x}'$ to be close to their counterpart on the spline $\hat{\mathbf{x}}'$. Figure 5.4 shows what we mean as the input points in **red** are incorrectly assigned to **blue** points on the spline as evidenced by the connecting **blue** lines. To counteract this, we sample the spline very finely and assign the input points to the new points $\check{\mathbf{x}}' \in \check{\mathbf{T}}'$ that are closest to them. In Figure 5.4, we show $\check{\mathbf{T}}'$ using **green** points and lines. Note how the **green** lines are much more often parallel to one another and perpendicular to the spline than the **blue** lines, meaning they approximate the input trajectory better.

As mentioned above, we now have an initialization for both the object shape $\mathbf{s}$ and its pose $\mathbf{p}_t$ at each time step t. We now want to modify these parameters to better explain our observed data, *i.e.* the video frames. We designed the following objective function:

$$E(\mathbf{s}, \mathbf{p}_t) = \lambda_T E_T + \lambda_F E_F + \lambda_A E_A + \lambda_P E_P + \lambda_S E_S, \tag{5.7}$$

where the λ balancing terms are experimentally set to the following default values

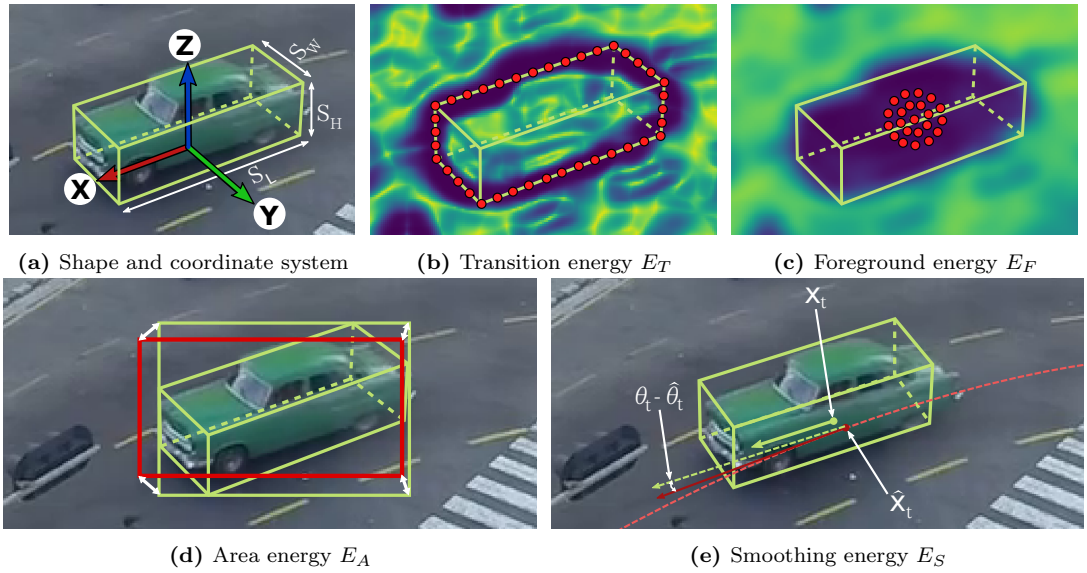
for all our experiments unless otherwise specified: $\lambda_T = \lambda_F = \lambda_P = 10$, $\lambda_S = 1$ and $\lambda_A = 2$. Each term in Equation 5.7 is designed to accomplish different goals and we show a visual representation of them in Figure 5.5. The transition E_T and foreground E_F energies work together to ensure the bounding volume fits well around the tracked object (see Figures 5.5b and 5.5c). We define the transition energy as

$$E_T(\mathbf{s}, \mathbf{p}_t) = \sum_{\mathbf{y} \in \mathcal{H}} \rho \left(e^{-\frac{|\nabla C_{FB}(\mathbf{y})|}{\sigma_T}} \right) \tag{5.8}$$

where $\mathbf{y}$ are locations of points regularly sampled on the convex hull $\mathcal{H}$ (shown in Figure 5.5b as **red** dots) of the object's bounding volume (defined by its shape $\mathbf{s}$ and pose $\mathbf{p}_t$ at time t) projected into image space. The Cauchy robust loss is defined as $\rho(s) = \log(1 + s)$ while we use $C_{FB}(\mathbf{y})$ to denote the foreground-to-background photo consistency at image-space location $\mathbf{y}$ and ∇C_{FB} its image gradients. For simplicity, we do not specify in which direction we compute the gradient of C_{FB} but, in our implementation, we define the numerator in the exponential as the sum between the absolute values of the gradient in both X and Y directions. Note that we bilinearly sample the discretized ∇C_{FB} at the continuous location $\mathbf{y}$ and we set $\sigma_T = 1$ for all our experiments. Intuitively and as seen in Figure 5.5b, the transition energy favors configurations where the edges of the bounding volume project to areas of transition between foreground and background. We define the foreground-to-background photo consistency C_{FB} as follows:

$$C_{FB}(\mathbf{x}) = \min \left(|\tilde{\mathtt{I}}_{\mathbf{x}} - \tilde{\mathtt{B}}_{\mathbf{x}}|, |\mathtt{I}'_{\mathbf{x}} - \mathtt{B}'_{\mathbf{x}}|, \tau_C \right) \tag{5.9}$$

where $\mathtt{I}$ and $\mathtt{B}$ are the ith input frame and the background median image respectively. $\mathtt{I}'$ and $\mathtt{B}'$ represent the respective images converted to gray from RGB and blurred using a Gaussian filter of size 11 with a variance of 10. Intuitively, these images contain the low frequency details captured by the camera. On the other hand, $\tilde{\mathtt{I}}$ and $\tilde{\mathtt{B}}$ are the original images from which we subtracted the low frequency details and therefore contain the high frequency information. Finally, the truncation threshold $\tau_C = 8$ is aimed at robustifying the measure against outliers. Intuitively, the higher the value of $C_{FB}(\mathbf{x})$, the bigger the appearance change between the color at a pixel in the input frame and the median background



(a) Shape and coordinate system **(b)** Transition energy E_T **(c)** Foreground energy E_F

(d) Area energy E_A **(e)** Smoothing energy E_S

Figure 5.5: Visual representations of our energy terms from Equation 5.7. (a) object coordinate system and sizes of cuboid shape $\mathbf{s}$. (b) the transition energy is computed at points shown as **red** dots which are regularly sampled on the convex hull of the projected bounding volume. (c) the foreground energy is computed at points shown as **red** dots which are regularly sampled on concentric circles starting from the center of the projected bounding volume; shades of **dark blue** represent low cost in both (b) and (c). (d) the area energy depends on the distance between matching corners (as marked by the white arrows) of the detected bounding box in **red** and the projected bounding volume's AABB in **green** (e) the smoothing energy depends on the distance between the estimated position $\mathbf{x}'_t$ and the predicted one $\hat{\mathbf{x}}'_t$ and the difference between the estimated orientation angle θ_t and the predicted one $\hat{\theta}_t$.

image, so it can be seen as a rough alpha matte (which we will use as such for rendering in Section 5.3.2) telling us where a moving object is. The foreground energy is defined as

$$E_F(\mathbf{s}, \mathbf{p}_t) = \sum_{\mathbf{y} \in \mathcal{R}} \rho \left(e^{-\frac{C_{FB}(\mathbf{y})}{\sigma_F}} \right) \tag{5.10}$$

where the image-space points $\mathbf{y}$ belong to rings $\mathcal{R}$ radiating from the center of the projected bounding volume (see **red** dots in Figure 5.5c) and we set $\sigma_F = 5$ for all our experiments. Intuitively, the foreground energy E_F favors configurations where the bounding volume encompasses foreground objects. The area energy E_A serves two purposes: i) it ensures that the bounding volume projects inside the

tracked 2D bounding box from Section 5.2.1 and ii) it favors cuboids that fit well within the given axis aligned boxes. We define

$$E_A(\mathbf{s}, \mathbf{p}_t) = \sum_{\mathbf{y}_\mathcal{O} \in \mathcal{O}_t,\, \mathbf{y}_\mathcal{C} \in \mathcal{C}_t} \rho\left(||\mathbf{y}_\mathcal{O} - \mathbf{y}_\mathcal{C}||\right) \tag{5.11}$$

where $\mathbf{y}_\mathcal{O}$ and $\mathbf{y}_\mathcal{C}$ are corresponding corners of the image-space, axis-aligned bounding boxes (AABB) $\mathcal{O}_t$ given by the object tracker (**red** rectangle in Figure 5.5d) and $\mathcal{C}_t$ encompassing the projected cuboid at time t respectively (**green** rectangle in Figure 5.5d). The distance between points $\mathbf{y}_\mathcal{O}$ and $\mathbf{y}_\mathcal{C}$ are shown as white arrows in Figure 5.5d. The shape prior energy E_P ensures that the estimated bounding volume does not collapse into flattened shapes. The prior consists of two Gaussian distributions that represent the ratios between the three sizes of the cuboid learned from object specific data. In our case, we computed the width-to-length and height-to-length ratios of the "Car", "Van" and "Truck" object types from the ground truth training data for the 2017 KITTI 3D Object Detection Evaluation benchmark [GLU12]. We then define E_P as

$$E_P(\mathbf{s}) = \sum_{m \in \{W,H\}} \rho\left(\frac{(s_m/s_L - \mu_m)^2}{2\sigma_m^2}\right) \tag{5.12}$$

where s_L, s_W and s_H are the length, width and height of the bounding volume respectively as defined in $\mathbf{s}$ and shown in Figure 5.5a. The parameters (μ_W, σ_W) and (μ_H, σ_H) are the parameters of the width-to-length ratio and height-to-length ratio Gaussians respectively. Finally, the smoothing energy E_S favors smooth trajectories by ensuring the position of the cuboid at each point in time does not diverge from the spline-smooth trajectory. We define

$$E_S(\mathbf{p}_t) = ||\mathbf{x}'_t - \hat{\mathbf{x}}'_t|| + |\theta_t - \hat{\theta}_t| \tag{5.13}$$

where $\mathbf{x}'_t$ and θ_t are the current estimates of position and orientation and $\hat{\mathbf{x}}'_t$ and $\hat{\theta}_t$ are the smoothed initialization. Figure 5.5e shows a visual example of the two terms on the right hand-side of Equation 5.13.

To ensure a better convergence rate, we have adopted a two step optimization process. In our experiments, optimizing all parameters $\{\mathbf{s}, \mathbf{p}_t, t \in \left[1, N\right]\}$ at once

proved problematic, likely due to bad initialization, the complexity of the energy function in Eq. 5.7 and the high dimensional search space. To counteract these issues, we first optimize a set of *global* parameters $\{\mathbf{s}, \mathbf{x}'_G, \theta_G, \mathbf{d}_G\}$, where $\mathbf{s}$ is the global shape of the bounding volume as described above, $\mathbf{x}'_G$ is the global (x, y) position of the trajectory on the ground plane, θ_G is a global rotation and $\mathbf{d}_G$ is a global dilation parameter which stretches the trajectory along the x and y-axes. We initialize $\mathbf{x}'_G = \hat{\mathbf{x}}'_0$, $\theta_G = \theta_0$ and $\mathbf{d}_G = \begin{bmatrix}1 & 1\end{bmatrix}$. We optimize the energy in Equation 5.7 using the Nelder-Mead algorithm [NM65] with $\lambda_S = 0$ as we are optimizing for a global transformation of the already smooth trajectory $\hat{\mathbf{T}}'$.

In the second optimization step, we *locally* refine each pose $\mathbf{p}_t$ at each time t separately. Here, we set $\lambda_P = 0$ as the shape $\mathbf{s}$ remains fixed for the duration of the local refinement. The aim of this step is to correct fine mistakes that occurred during smoothing and cannot be solved by the global transformation found during the first step. After local refinement, we smooth the resulting trajectory $\mathbf{T}'$ once again as adjusting the poses separately may introduce noise due to occlusions. The smoothed trajectory is fed through the global optimization once again and we repeat for 10 iterations.

5.3 Applications

At the beginning of this chapter, we have anticipated that the reason we have decided to undertake the task of tracking object shapes in 3D given 2D tracks is that it opens up a number of interesting applications. In the following sections we will describe the ones we have had a try at.

5.3.1 From Tracked Cuboids to Textured Models

Given the tracked cuboids described in Section 5.2 and the camera calibration from Section 5.1, we can reason about the way the appearance of the tracked object changes over time as it changes position and orientation w.r.t. the static camera. *Multi-view stereo* (MVS) algorithms aim to reconstruct the 3D shape of static scenes or objects given a set of images and the poses of the camera when it captured them [FH*15]. In typical MVS scenarios, the images are captured

by the same camera over time as it moves in the scene. Since such algorithms rely on finding point correspondences based on appearance, it is critical that the scene does not change drastically. While the objects in our case are moving and changing appearance over time, the camera is completely static, so if we reverse the way we look at our data and consider the objects as being static and the camera to be moving w.r.t. them, we can employ standard MVS reasoning to reconstruct the shape and appearance of our filmed objects.

In this section, we aim to show that the tracks we recover in Section 5.2 are of high quality enough to be usable in the context of 3D reconstruction. This in turn has the potential to enable novel interactions with video content which we explore in the next Section 5.3.2. As MVS is still a very active research field [FH*15], more recent and accurate methods are likely to produce better results. However, we believe our choice of a reasonably simple but, crucially, easy to implement method, is enough to show the usefulness of our tracking method.

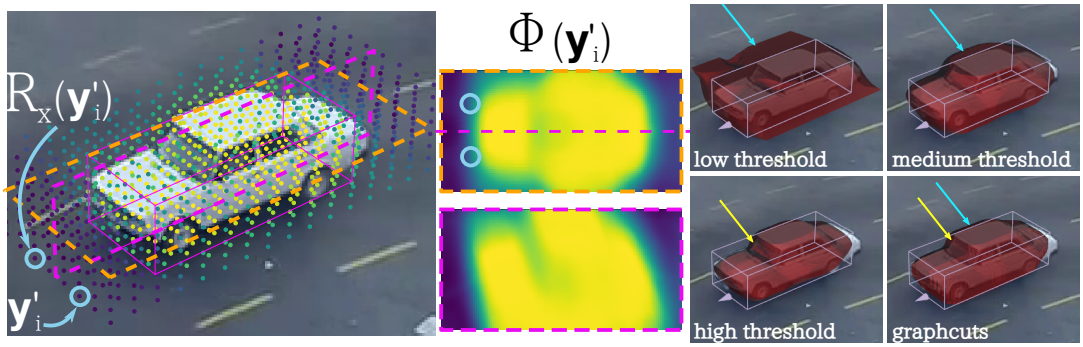
Our chosen 3D reconstruction algorithm belongs to the *Volumetric Data Fusion* family. We regularly sample the estimated cuboid volume to yield a set of 3D points $\mathcal{V}(\mathbf{s}) = \{\mathbf{y}'_i\}$. We define image-space points

$$\mathbf{y}_{ti} = P(\mathbf{p}_t, \mathbf{y}'_i), \tag{5.14}$$

where P projects the ith 3D point $\mathbf{y}'_i$ (see Figure 5.6a), as shown in Equation 5.1, after placing it into world-space according to the cuboid's pose $\mathbf{p}_t$ at time t. We then accumulate the photo consistency C from Equation 5.9 at each voxel over time:

$$\Phi(\mathbf{y}'_i) = \sum_t w_t(C(\mathbf{y}_{ti}) + C(R_x(\mathbf{y}_{ti}))), \tag{5.15}$$

where w_t is a weight defined in Equation 5.19 that signifies how much we trust frame t and R_x returns the location of voxel $\mathbf{y}_{ti}$ reflected about the X axis, which ensures symmetry as shown in Figure 5.6a. Intuitively, $\Phi(\mathbf{y}'_i)$ returns a high value whenever the voxel $\mathbf{y}'_i$ mostly projects to areas of high photo consistency over time, meaning they belong to the foreground tracked object. We could now threshold the values of Φ across the voxel grid and recover the shape of the object by triangulating it using *Marching Cubes* [LLVT03]. However, the



(a) Cost volume **(b)** Reconstruction quality

Figure 5.6: Visual representation of the reconstruction cost and final quality. (a) We sample the volume in and around the given object bounding volume (**magenta** outline) at regular intervals, shown as colored dots. The color coding signifies the probability that a voxel is part of the tracked object where shades of yellow denote higher values. We also show **sagittal** and **horizontal** slices of the cost volume $\Phi(\mathbf{y}'_i)$. We ensure symmetry along the **magenta** saggital plane by summing over the photo consistency values of points $\mathbf{y}'_i$ and their counterparts reflected along the plane $R_x(\mathbf{y}'_i)$. (b) The quality of the recovered geometry is unpredictable if we simply threshold the values of $\Phi(\mathbf{y}'_i)$. Meshes can be too big as shown by **cyan** arrows, or too small and with holes as shown for the car's windshield highlighted by yellow arrows. Using graphcuts we can avoid having to select a threshold value and results are more consistent with the image data.

photo consistency measure tends to be noisy whenever the tracked object has similar colors to the background, so a manually chosen threshold may either introduce holes or estimate an inaccurately enlarged shape as evidenced by the examples in Figure 5.6b. Moreover, the ideal threshold value may be different depending on the object or even depending on the voxel, making choosing it manually impractical. One workaround found in the literature is to use 3D graph cuts [BVZ99] on the voxel grid to ensure smoothness. To this aim, we introduce the smoothness pairwise term between neighboring voxels $\mathbf{y}'_i$ and $\mathbf{y}'_j$:

$$\Psi(\mathbf{y}'_i, \mathbf{y}'_j) = \min\left[0.05, \sum_t w_t \frac{C_I(\mathbf{p}_t, \mathbf{m}'_{ij}) + C_I(\mathbf{p}_t, R_x(\mathbf{m}'_{ij}))}{2}\right], \tag{5.16}$$

where $\mathbf{m}'_{ij} = \frac{\mathbf{y}'_i - \mathbf{y}'_j}{2}$ is the middle point between $\mathbf{y}'_i$ and $\mathbf{y}'_j$ and $R_x(\mathbf{m}'_{ij})$ is the position of $\mathbf{m}'_{ij}$ reflected about the X axis as before. The image color photo

consistency

$$C_I(\mathbf{p}_t, \mathbf{x}') = |I(P(\mathbf{p}_t, \mathbf{x}')) - I_\mu(\mathbf{x}')|$$

penalizes 3D points $\mathbf{x}'$ where the color changes over time so $I(\mathbf{x}_t) \in \left[0, 1\right]$ denotes the color at image-space location $\mathbf{x}_t$ while $I_\mu(\mathbf{x}') \in \left[0, 1\right]$ indicates the mean color at the image-space locations where the 3D point $\mathbf{x}'$ projects to over time. We then use standard binary Graph Cuts [BVZ99] to minimize the objective function:

$$E(\{\mathbf{y}_i'\}) = \sum_i \Phi(\mathbf{y}_i') + \sum_{i,j \in \mathcal{N}} \Psi(\mathbf{y}_i', \mathbf{y}_j'). \tag{5.17}$$

Minimizing the above energy yields a binary segmentation of the 3D voxel grid into foreground and background voxels which we then triangulate using [LLVT03]. We show sample results in Figure 5.9 and on our website[1].

Defining w_t in Equations 5.15 and 5.16. Given the triangular mesh resulting from [LLVT03] as a set of vertices $\mathcal{M} = \{\mathbf{v}_i', i \in \left[1, M\right]\}$, we compute a per vertex color texture $\mathcal{T}$ as following:

$$\mathcal{T}(\mathbf{v}_i') = \sum_t w_t \frac{I(\mathbf{v}_{ti}) + I(R_x(\mathbf{v}_{ti}))}{2} \tag{5.18}$$

where $\mathbf{v_{ti}}$ is defined as in Equation 5.14 and $R_x(\mathbf{v}_{ti})$ denotes $\mathbf{v_{ti}}$ reflected about the X axis. While initially $w_t = 1/N$ where N is the number of frames, we compute a new value for w_t after computing $\mathcal{T}$ as

$$w_t = e^{-\frac{M^{-1} \sum_i \min\left[0.1,\, |\mathcal{T}(\mathbf{v}_i') - I(\mathbf{v}_{ti})|\right]}{\sigma_w}} \tag{5.19}$$

where we set $\sigma_w = 0.0075$ for all our experiments. Given the new weights w_t, we now iterate the triangulation-texturing-reweighting loop until convergence which in our experiments is 6 iterations. Intuitively, w_t will down-weight frames that do not agree well with the reconstructed model which is due to noise in the tracked cuboid.

[1] https://corneliu.co.uk/phdresults/chapter5/index.html

5.3.2 Video-Based Rendering

Thanks to the techniques described above, we now have a fully calibrated camera setup (*i.e.* intrinsics and extrinsics in Equations 5.2 and 5.3), a reconstructed three-dimensional textured mesh per tracked object, as described in Section 5.3.1 and their pose in the 3D world as shown in Section 5.2. By extending our 3D viewer described in Section 3.6.1 we can now synthesize even more compelling 3D visuals from a monocular static camera video.

Briefly, image-based rendering (IBR) techniques aim to produce photo-realistic 3D visuals interactively (*e.g.* by giving users control over the virtual camera) by leveraging the information captured through traditional photography. Typically, this is done by reconstructing the scene geometry through methods such as MVS [FH*15] and use it to decide which color taken from the input images to display at each 3D world point. In this section, we show that we can adopt the same mindset. We did not aim to innovate in the field of IBR, so we here only describe what is necessary to replicate our results. For more information on IBR, we defer the reader to [SCK08].

For our demonstration purposes we adapted our 3D viewer from Section 3.6.1 to render multiple tracked objects, either using textured meshes or Unstructured Lumigraph Rendering (ULR) [BBM*01] (see Fig. 5.7). We chose ULR because it has been shown to produce great quality visuals in constrained scenarios while still being relatively simple and, crucially, fast. In a few words, ULR synthesizes a new view of a 3D scene by casting rays from the center of projection through each pixel. The color at each point hit by rays is set by alpha blending between the colors seen at those points in multiple input images. The alpha blending weights are set according to how close the requested view matches the input views. Intuitively, the appearance of a point depends on its appearance in the input photos and, if we are looking at it in the new virtual view from a similar viewpoint as one of the input views, its appearance should remain the same. Formally, the color $\mathtt{O}_{\mathbf{s}}$ at a pixel location $\mathbf{s}$ in the output image $\mathtt{O}$ is defined as the weighted sum

$$\mathtt{O}_{\mathbf{s}} = \sum_{i=1}^{N} w_i \mathtt{I}_{\mathbf{s}_i} \tag{5.20}$$

(a) Top-down view **(b)** Off-trajectory pose

Figure 5.7: Free-viewpoint interactive video. (a) Using our real-time 3D viewer, we can move the virtual camera to see the scene from any view, such as the shown top-down view. Compare with the very different cropped original view point in the inset. Users can show all the tracked objects with their trajectories, bounding boxes and textured meshes and even control them manually to move them away from the originally captured trajectory. (b) Close-up view of one tracked car at a new location away from the original trajectory (**green** curve). Note the difference in quality between using a textured model (top) and the ULR algorithm (bottom). The details are much sharper and the over-inflated geometry is hidden. Please see videos on our website mentioned in our results section 5.4.2 for demo videos showing view-dependent effects such as specularities.

where $\mathtt{I}_{\mathbf{s}_i}$ represents the color of the pixel location $\mathbf{s}_i$ in the ith input image $\mathtt{I}$ and the weight $w_i = \exp\left[-(\mathbf{y}' \rightarrow \mathbf{s}' \rightarrow \mathbf{y}'_i)/k_C\right]$ is the angle between the vector defined by the world-space points $\mathbf{y}'$ and $\mathbf{s}'$ and the one defined by $\mathbf{y}'_i$ and $\mathbf{s}'$. Here, $\mathbf{s}'$ represents the world-space point that projects to pixel location $\mathbf{s}$ in the desired view and $\mathbf{s}_i$ in the ith view, while $\mathbf{y}'$ and $\mathbf{y}'_i$ are the world-space locations of the desired and ith cameras respectively. Intuitively, the bigger the angle between the two vectors, the smaller the weight and that specific input camera contributes less to the final color seen at a pixel. The parameter k_C is user-tunable and the larger, the more forgiving we are about whether the desired view matches a particular input view. At each render call, we select the best $N = 4$ input views to blend between based on how close they match the desired view. Note that, in our results we also vary the alpha value of a 3D point based on the foreground-to-background

Name	*# frames*	*# objects*	*Max obj per frame*	*Average track length*	*Used detections %*
HAVANA	5400	86	9	168	43.2
JACKSON	5050	52	6	256	45.8
ABBEYROAD	3000	32	5	203	61.0
SIMPLECROSSING	274	5	3	68	25.8
MVI40243	1265	165	17	81	64.5
MVI40732	2120	41	8	73	36.5
AUBURN	2773	33	5	251	34.6
GRANGE	2860	75	5	127	26.1
ROSS	2410	62	8	128	53.1
MVI40871	1720	77	15	123	20.2

Table 5.1: Our tracking and 3D reconstruction datasets. For each video we provide the number of frames, the number of tracked objects, the max number of objects in a single frame, the average track length (in frames) and the percentage of used detections. The detections found by [WLW*17] can be culled by the tracker [BES17] and we remove all the tracked objects that are stationary and the frames of moving objects where they are partially outside of the video frame.

photo consistency from Equation 5.9 which acts as an alpha mask. Figure 5.7b shows the difference between simply using a textured model and the much higher quality ULR result.

5.4 Results

Using the algorithms discussed in this chapter, we have processed a number of input videos. We summarize them in Table 5.1 and present qualitative results in Figures 5.8 and 5.9 and on our website[1]. We have found on the Internet and processed with our pipeline 10 videos at varying resolutions, view points and car density. We summarize our datasets in Table 5.1 where we provide various statistics such as the total number of frames or the maximum object density in a frame. All videos depict traffic intersections or similar scenarios where there are plenty of moving vehicles that we can detect using [WLW*17] and track

[1] https://corneliu.co.uk/phdresults/chapter5/index.html

using [BES17]. We have chosen the videos to highlight the robustness of our method to various scenarios:

1. **Viewpoint**: Havana and SimpleCrossing showcase top-down like viewpoints, whereas AbbeyRoad and Jackson present street-level viewpoints and, as such, objects suffer from heavy occlusions at times;

2. **Object density**: the amount of objects present on screen at once varies widely with SimpleCrossing only containing a few objects that barely interact with one another, while Havana and MVI40243 contain tens of objects at once;

3. **Video quality**: AbbeyRoad and Ross are low resolution videos while Havana and Jackson are HD videos. Moreover, there are various amounts of noise and compression artefacts in live-streaming webcameras such as Jackson and Grange while AbbeyRoad and Ross contain skipped or repeated frames;

4. **Miscellaneous**: other challenging factors include motion blur due to fast moving vehicles (MVI40243), non-planar ground (MVI40732), extreme weather conditions (*e.g.* rain in Auburn) and heavily cluttered background where a clean median image cannot be obtained (MVI40871).

We now discuss in more details the results for our two scenarios. In Section 5.4.1, we present our tracking results and the ways our algorithm is consistently underperforming in our wide range of videos described in Table 5.1. In Section 5.4.2 we present our 3D reconstruction and 3D VBR results and discuss the interactive video experience that they enable.

5.4.1 Tracking

Visually inspecting the result videos [1] reveals that our tracking results are generally very stable and the estimated bounding volume sizes encompass the tracked cars very tightly. Our goal in this chapter was not to necessarily improve on object tracking but rather augment the two-dimensional information provided

[1] https://corneliu.co.uk/phdresults/chapter5/index.html

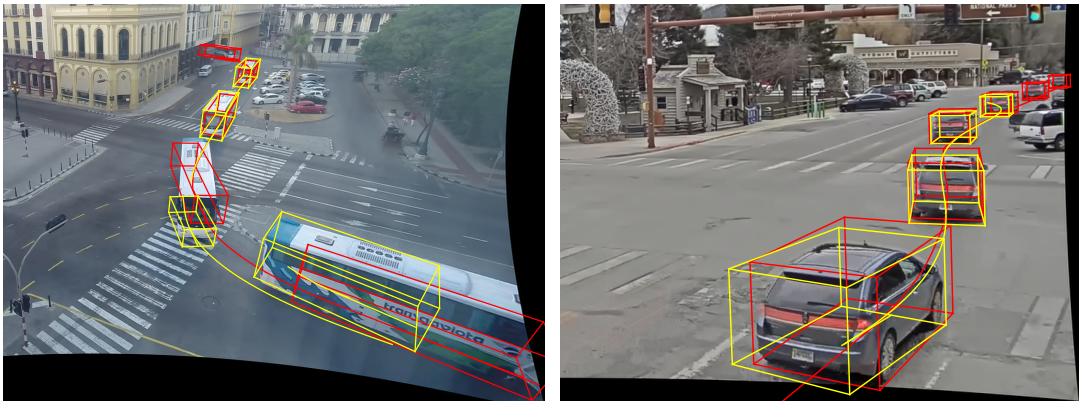
Name	*# frames*	*# tracks*	*3D IoU*	*Missing %*	*Time [m]*
RedCar	489	1	0.755	4.3	143
BlueCar	274	1	0.714	18.6	82
GreyCar	792	1	0.515	60.2	210
GreenCar	264	1	0.787	24.6	79
BlackTruck	251	2	0.304	32.3	106
WhiteBus	1270	3	0.438	26.7	342

Table 5.2: Tracking accuracy of our automatic algorithm compared to manually tracked objects in a variety of input videos. From left to right: name of the sequence, number of manually tracked frames, number of automatically found tracks for each object, intersection over union (closer to 1 is better), percentage of frames that have not been tracked automatically (due to the detector [WLW*17] not detecting the object), time in minutes necessary to manually track each object.

by traditional trackers with additional data that can enable more compelling video interactions such as described in Section 5.3. For this reason, we do not evaluate on traditional 2D multi-object tracking benchmarks such as [MLTR*16]. Instead, we show that the augmentations we automatically infer, *i.e.* the size of the bounding volume and its pose over time, are accurate w.r.t. manually annotated sequences of frames. In Table 5.2 we present 6 sequences in which we manually tracked a bounding volume following a given car. Note how the user effort in terms of minutes spent tracking an object manually is considerable while our method is completely automatic and leaves the user time to focus on the creative side instead. The 3D Intersection-over-Union (IoU) measure shown in Table 5.2 is defined as follows

$$\text{3DIoU} = \frac{1}{N}\sum_{i=1}^{N}\frac{\text{area}(F_i \cap F_i')}{\text{area}(F_i \cup F_i')}\frac{\min(H_i, H_i')}{\max(H_i, H_i')} \tag{5.21}$$

where F_i and F_i' are the automatically and manually defined footprints in the ith frame respectively and similarly H_i and H_i' are the heights of the object. We only compute the 3D IoU for the frames where the object has been detected by the car detector [WLW*17]. Multiple automatic tracks may correspond to the same manually tracked object as the tracker we have chosen [BES17] splits tracks if



(a) WhiteBus **(b)** GreyCar

Figure 5.8: Manual vs automatic bounding volume tracking. In **red** we show the manually defined trajectory and oriented 3D cuboid, while in **yellow** we show the automatically inferred ones using our algorithm from Section 5.2. Note that some objects have been split into multiple individual tracks as a result of the tracker [BES17]. Also, note how the manually defined tracks may not approximate well the shape and pose of the object (especially in (a)) despite spending hours manually tuning the result (see Table 5.2) due to the inherent difficulty of 3D tracking in image space. The full videos for all the evaluated sequences can be seen on our website.

the object is not detected in subsequent frames. Note that the WhiteBus object has a very low IoU score for two reason: i) the automatic tracker has mistakenly assigned a small bounding volume to the object, likely due to a combination of bad detection 2D boxes and the shape prior's (Eq. 5.12) inability to represent the shape of a bus and ii) the fact that the manual track does not represent the bus well as seen in Figure 5.8a and result videos [1]. We have chosen to show this faulty track nonetheless as it clearly shows that despite large user effort (more than 4 hours in Table 5.2) the results can still be not satisfactory and the need for automatic, robust and accurate algorithms, such as presented in this chapter, becomes evident.

Qualitatively speaking, the result videos highlight the two ways our tracker consistently under-performs. First, a number of frames at the beginning or at the end of an object's track may not represent the trajectory accurately. This does not influence the reconstruction of the object's 3D shape thanks to our iterative

[1] https://corneliu.co.uk/phdresults/chapter5/index.html

process described in Section 5.3.1 which dynamically down-weighs frames that do not agree with the estimated textured mesh. It does however introduce artefacts when performing ULR as the incorrect pose of the object results in inaccurate color blending (as we discuss in Section 5.4.2). Second, the scale of an object can be incorrect and may not encompass the object tightly when projected into image space especially if i) it is found far away from the camera or at grazing angles (*e.g.* object 178 in HAVANA in Figure 5.10c and on our website[1]) or ii) it moves parallel to the camera due to the inherent ambiguity in our photo consistency costs (*e.g.* object 51 in GRANGE in Figure 5.10b and on our website[2]). We discuss reasons and potential solutions in the Limitations and Future Work section (5.5.1).

5.4.2 Interactive 3D Video Experiences

After tracking vehicles in the videos described in Table 5.1, we reconstructed the shape of each object using the algorithm presented in Section 5.3.1. We show the estimated mesh both with and without texture in Figure 5.9 and on our website[3] along with sample cropped video frames. Qualitatively, the recovered meshes present artefacts due to the discretization of the object volume into a grid of voxels and the graph cut-based segmentation described in Section 5.3.1 (see Fig. 5.9). Moreover, the quality of the meshes vastly depends on the number of viewpoints and their diversity. In extreme cases, such as the third row in Figure 5.9 and 5.10g, they present typical voxel carving artefacts, such as an elongated shape in the directions that are not clearly visible in the input video. Finally, reconstructed meshes degenerate when the tracking has failed or objects look similar to the background colors as shown in Figure 5.10f. This is due to the fact that the carving cost (Eq. 5.15), used to separate foreground voxels from background ones, depends on photo consistency (Eq. 5.9).

Despite the arguably mediocre reconstructions shown in Figure 5.9, the ULR technique we describe in Section 5.3.2 is successful at rendering tracked objects from novel view points. As we show in our result videos[4], we allow users to interact

[1] https://corneliu.co.uk/phdresults/chapter5/havana/tracking.html
[2] https://corneliu.co.uk/phdresults/chapter5/grange/tracking.html
[3] https://corneliu.co.uk/phdresults/chapter5/index.html
[4] https://corneliu.co.uk/phdresults/chapter5/index.html

Figure 5.9: Reconstruction and 3D VBR results. From left to right: sample cropped input frame, front mesh, front textured mesh, back mesh, back textured mesh, ULR example. Please see our supplemental website for more results.

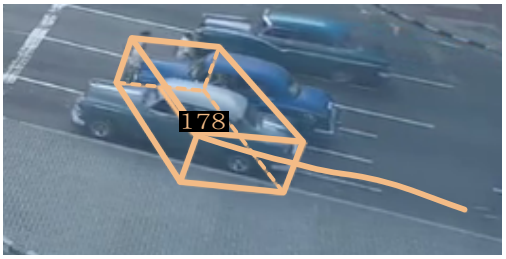
with objects filmed by a static camera in very compelling ways. Thanks to the algorithms presented in this chapter, we can now turn videos into interactive playgrounds, where users are given the ability to not only change the speed at which cars move (such as is possible in our Counter Loop game prototype from Section 4.6.1) but also make them follow a different lane to the one they were filmed following and even change the trajectory completely. We show this ability in a variety of videos as a testament of the fact that videos can become fully interactive media. Users are no longer limited to passively consuming the recorded content and can actively influence what happens on screen in the intuitive ways

previously only possible in video games.

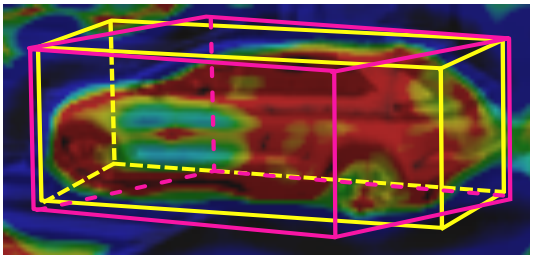
5.5 Conclusions

In this chapter, we presented novel algorithms for inferring 3D information about the real world from monocular static camera videos. In Section 5.1 we show that simply using the location of the horizon line in an image is enough to estimate the location of a ground plane and the camera's pose w.r.t. it. Finding the horizon line is a much easier problem [WZJ16] than a set of orthogonal vanishing points, especially when the Manhattan-world assumption is broken, which is traditionally used for single view camera pose estimation such as seen in [OSGO12]. We also describe a manual tool which requires very little user input to estimate camera pose and even reconstruct a rough mesh for representing the mostly static background present in static-camera videos.

We then use the camera parameters and an off-the-shelf multi-object tracker to estimate object shape and pose over time (Sec. 5.2) as they move on the ground plane and use it to recover "good enough" meshes (Sec. 5.3.1) to use for photo-realistic novel-view rendering (Sec. 5.3.2). All these elements come together to allow one to interact with videos in new and compelling ways, as users are given the ability to move the input camera to see the scene from a new viewpoint and even directly control how objects move across the world. In our effort to create compelling interactive video experiences, we have shown that traditional filmed content can be enjoyed in a game-like scenario where users have full control over what they see. Additionally, we have made great strides towards enabling ever more compelling and intuitive ways of interacting with video content without requiring prohibitive user effort or even none at all for processing it. While automatisms have been introduced in this chapter, we still believe users should be given control over both quality-related issues and creative decisions and we now discuss them.



(a) The cuboid pose may be inaccurate at trajectory extremities as the spline over-fits to noisy estimates due to other nearby foreground objects.

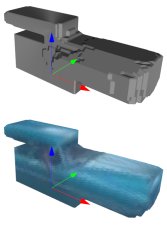


(b) Multiple cuboids may correspond to a low (darker colors) foreground-to-background cost if the object moves parallel to the camera.

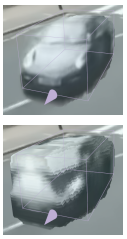
(c) Bad calibration may lead to an estimated bounding volume that fits the object well at mid range but not when far away (box too big) or at grazing angles (box too small).

(d) Mixing colors from views where object is occluded, results in artefacts.

(e) Ghosting due to misalignment.



(f) Bad mesh due to similar background.



(g) ULR (top) hides bad carve (bottom).

Figure 5.10: We show a number of limitations of the algorithms presented in this chapter. Please see the text for more details.

5.5.1 Limitations and future work

In Section 5.4 we presented and discussed our results along with mentioning a number of issues manifesting themselves at both the tracking and the reconstruction level. In terms of tracking accuracy, there are two prominent issues. First, the recovered tracks are noisy at the extremes and exhibit unpredictable behavior that does not well match the tracked objects. We believe this is mainly due to poor photo consistency quality, which is largely affected by occlusions and similar appearance of the object to the background. This manifests itself as noise in the pose of the object at the local refinement stage where the spline-based smoothing we employ fails to recover the clean trajectory typical of moving vehicles (see Fig. 5.10a). While our photo consistency measure is defined in terms of appearance

difference between a given video frame and the static background, a more accurate definition would be in terms of the appearance of the tracked object itself. A future extension could attempt to learn a model of appearance that accurately describes a given object [CET01] which would give a more consistent measure of photo consistency as nearby foreground objects would be ignored despite them not being background. Another future direction to investigate is to introduce temporal consistency into the photo consistency computation as by definition the appearance of an object would not change drastically in subsequent frames.

The second issue with the tracker's performance, concerns mistaken estimated object shape which may happen due to the way we estimate camera pose. First, if the estimated plane is not well aligned with the real world, the projection of the bounding volume is inaccurate at grazing angles and far away from the camera. This results in the issues highlighted in Figure 5.10c where the estimated bounding volume fits the vehicle well at mid range but it does not when it is far away or close to the camera. Second, similar artefacts can be seen when our assumption that the ground on which the objects move is planar such as in our MVI40871 dataset[1]. These issues could be addressed by iteratively refining the estimate of the ground plane to better fit the movement of objects as part of our optimization from Section 5.2.2. Another option would be to not assume the ground is completely planar and instead fit a triangular mesh to the initial plane and manipulate its vertices to better explain the way the tracked objects move. A different reason for incorrect object shape is an inherent ambiguity in our foreground-to-background photo consistency cost in Equation 5.10. As seen in Figure 5.10b, if an object moves parallel to the camera, there are multiple low-cost solutions and the optimizer may get stuck in local minima.

In terms of reconstructing the tracked objects and rendering them from novel viewpoints, the algorithms we described and employed are rather simplistic. We believe they demonstrate our proof of concept use-case very well but could be improved on. Better reconstruction could be achieved by improving on the quality of the photo consistency measure as mentioned above. The shape would better fit the object when projected into the image space and there would be less degenerate meshes (see Fig. 5.10f) as objects would be distinguishable from the background

[1] https://corneliu.co.uk/phdresults/chapter5/MVI_40871/tracking.html

even if they have similar appearance. Moreover, having better photo consistency would improve the quality of the tracks which in turn would result in more views aligning well with the reconstructed shape. Our iterative optimization from Section 5.3.1 would thus be able to use more viewpoints for carving the voxel volume. Finally, better alignment between the object, the tracked volume and the reconstructed shape would result in better novel-view rendering as more views can be blended together without ghosting artefacts (see Figure 5.10e). Rendering could be improved further by dynamically choosing the best views based on how well they match the reconstructed mesh and by leveraging its appearance to detect occluders (*e.g.* see the light pole in HAVANA which gets blended with a car's appearance in Figure 5.10d) and inpainting them away.

A future direction to be researched is related to addressing one of the main limitation of the 2D box tracker we have chosen. The IoU tracker [BES17] detects objects simply by grouping object detections in subsequent frames. If the detector skips even one frame because of, for instance, the object getting partially occluded, the tracker will create two separate tracks for the same real-world object. This can be seen in our HAVANA dataset[1] where objects 232 and 249 refer to the same car. We could address this by using the estimated track to predict an object's pose in frames it was not tracked in and merge tracks if they are predicted to occupy the same space. Finally, the same reasoning can be used to extend a track beyond what was tracked using [BES17] and use our local refinement optimization to correct inaccurate predictions which would address the current limitation of discarding frames where an object is partially outside the video frame.

[1] https://corneliu.co.uk/phdresults/chapter5/havana/tracking.html

Chapter 6

Conclusion

In this thesis, we explored issues and bottlenecks that prevent content creators from converting videos into interactive experiences. Traditionally, videos are consumed passively by viewers who can simply watch what the creator envisioned. In contrast, video games are a highly interactive medium where players are in full control over how they enjoy the experience. We hypothesized that content creators are willing to spend the time and effort to make watching videos a more game-like experience. This is a hard problem for several reasons. First, filmed dynamic events can vary widely so identifying desirable and meaningful ways to interact with them becomes problematic. Second, the subjects of a video can be very different and their appearance can vary drastically over time. It therefore becomes crucial to design algorithms that are generic and robust enough to cope with such variations. Finally, fully automatic algorithms can become brittle so it is important to design the tools that integrate them to enable users to correct and improve upon results in order to reach the quality levels they seek. At the same time, automatisms should help users reach their goals efficiently while not hindering the creative process as, ultimately, it is the creator that knows how the experience should engage users.

We started this thesis by asking how videos could become more engaging and go beyond simple infinite seamless playback. In Chapter 3 we focus on the engagement factor and present an in-depth exploration of modes of interaction between users and filmed content along with the techniques necessary to make them reality. We identify video looping as seen in Video Textures by Schödl *et*

al. [SSSE00] as the core idea behind making videos into interactive experiences. We then explore what has prevented loopable videos from actively being used by content creators in practical settings. Lack of control over the synthesized output and the fact that mistakes cannot be easily corrected were identified as the most important limitations. The chapter goes on to present novel ideas and algorithms that are designed to overcome these limitations, such as our *appearance-based semantic action* grouping of video frames which can be used to influence the output video in meaningful ways. Our discussions with content creators (*e.g.* video game developers) also highlighted the need for efficient authoring tools so we place great emphasis on enabling users to preview and manipulate the results of automatic algorithms.

The concepts and techniques we envisioned come together in the *end-to-end system* we present in Chapter 4. Our tool is designed to support content creators as they convert videos into a library of loopable elements. They can then be combined together and controlled in real time as part of our new medium of expression: the *live video performance*. On-screen elements instantly react in meaningful ways to simple and intuitive end-user requests. For the first time, videos are not consumed in a fully passive way and instead, people can actively influence what happens in the video and when. Finally, a number of artists and end-users helped us evaluate our novel system and how our new interaction paradigm can be used for authoring of videos.

A further step was taken towards understanding the type of information one can infer from video to enable real-time interactivity. In Chapter 5 we explored the ability to manipulate video content in 3D. Our discussions with game developers highlighted that modern video games have become more appealing to the masses thanks to their ability to emulate the three-dimensional real world. Our end-to-end system from Chapter 4 can only manipulate pixels in image space so end-user can only interact with the experience in a two-dimensional manner. However, filmed events and subjects are recorded interacting with a three-dimensional world so enabling end-users to interact with them in 3D would result in a more compelling and engaging experience. We dedicated Chapter 5 to presenting a 2D-to-3D tracker that enables new applications such as reconstructing the three-dimensional shape of filmed objects and viewing them from novel viewpoints that were not

filmed in the input video.

Our original hypothesis stated that interactive tools and automation can assist content creators in the creation of video experiences to engage audiences in new ways. Throughout the thesis, it became evident that different videos vary greatly, both in how they are captured and the nature of their content. Because of this, the way videos are processed and the modes of interactions designed to engage audiences are tightly interconnected. Assumptions must inevitably be made to cope with input variation; this, in turn, reduces the potential impact of a tool or technique because generalisation is traded for specialisation. For instance, not all videos capture actors performing actions. In such cases, the methods described in Chapter 4 would not be appropriate. Although these assumptions seem limiting at first, they perform the vital role of constraining a problem to make it tractable. Thinking about rapid prototyping in the manner described by the artists in Section 4.8 would perhaps not be possible without first limiting the video domain.

The techniques described in this thesis make video content reactive to user input. However, bringing these experiences to broader audiences still poses great challenges. From the perspective of professional users, such as visual effect artists, robustness and particularly visual quality are paramount. On the other hand, casual users may be more interested in the final product (*e.g.* using the MakeyMakey as shown in Chapter 4 or playing a driving game built with techniques described in Chapter 5), so designing fun and engaging interactions is more important. As mentioned above, making assumptions on characteristics exhibited by input videos makes a problem tractable but limits use cases. However, these assumption can be interchanged or combined to make new use cases become apparent, possibly opening the door to more varied modes of interaction.

6.1 Possible Future Directions

In this section, we present a number of possible future research directions that improve upon and extend the topics discussed in this thesis. We discuss both technical improvements that are likely to increase quality and robustness as well as significant extensions that may enable ever more compelling interactive video

experiences.

Throughout this thesis, we have made an important assumption about the input data: they are monocular, static-camera videos. On the one hand, this limits our use cases but on the other hand we do not require specific capture setups such as multi-camera rigs or depth sensors so any casually captured video on the Internet is potentially usable. One interesting research direction is to do away with such an assumption and leverage the extra information in moving-camera videos for more immersive experiences. If the camera is moving, we could employ 3D reasoning on both filmed subjects and on the background scene such as done in MVS pipelines.

The system we describe in Chapter 4 provides the tools necessary to interactively prepare footage for live video performances. While we made the preparation process as streamlined and efficient as possible, some sequences such as HAVANA still require considerable user effort because of the sheer amount of moving elements and their complex interactions with one another. One interesting avenue for research would be to employ the reasoning described in Chapter 5 to more efficiently assist users in accurately tracking and segmenting moving elements. This would require more effort in adapting the algorithms we described to be more efficient and suitable for a user-in-the-loop setting.

In Chapter 4 we described a novel way to intuitively influence video synthesis by leveraging essentially semantic knowledge that users can easily define and interpret. The interactions described in Chapter 5 on the other hand, are more traditional as they replicate what is usually done in video games, *i.e.* moving the virtual camera and revealing new views of interactive objects by moving them across the scene. It would be very interesting to combine the two types of interactions for ever more engaging experiences. For instance, we could film a tennis match, track the players using the techniques from Chapter 5 and assign semantic knowledge to their movement using the system from Chpater 4. We could then enable players to move the players across the tennis court and requesting them to perform specific shots that were filmed in the input video. Moreover, 3D information could be used to define better looping in a similar fashion to character animation techniques such as [PKC*16, CVCH14] or more meaningful actions and the compatibility between them. For instance, we could define compatibility in

such a way that the second player would always counteract what the first player does. This may result in the second player going to the correct place on the tennis court to hit the ball at the right time and present a challenge for the human player.

Finally, there are multiple research directions that could improve upon and expand our results in Chapter 5. First, it would be interesting to do away with our assumption that tracked objects are mostly rigid and relatively well explained by cuboids. Moreover, the tracking (as is or for non-rigid shapes) would benefit from better photo consistency measures as discussed in Section 5.5 but also if we were to add a motion model to the equation. A future in-depth performance evaluation of the numerous steps in our pipeline may shed light on suitable extensions. Performance may improve if tracking and reconstruction were to be performed jointly in a similar manner to what is seen in [LHB15], although feature-based tracking may not perform well on our datasets as the moving objects are typically below 1% of the whole frame area. Finally, in Section 5.3.2 we show that we can synthesize novel views of the scene and the filmed moving objects. Our implementation is a proof of concept aimed at demonstrating that our contributions enable this new way of interacting with videos. The rendering quality could be improved upon by researching new ways of optimizing tracking and reconstruction for the purpose of minimizing visual artefacts such as ghosting.

References

[AF02] Arikan O., Forsyth D. A.: Interactive motion generation from examples. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 483–490. 15, 40, 47

[ASS*12] Achanta R., Shaji A., Smith K., Lucchi A., Fua P., Süsstrunk S.: Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence 34*, 11 (2012), 2274–2282. 53

[AZP*05] Agarwala A., Zheng K. C., Pal C., Agrawala M., Cohen M., Curless B., Salesin D., Szeliski R.: Panoramic video textures. In *ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), SIGGRAPH '05, ACM, pp. 821–827. 9

[BAAR12] Bai J., Agarwala A., Agrawala M., Ramamoorthi R.: Selectively de-animating video. *ACM Transactions on Graphics* (2012). 10, 11, 12, 22, 38

[BAAR13] Bai J., Agarwala A., Agrawala M., Ramamoorthi R.: Automatic cinemagraph portraits. *Computer Graphics Forum (EGSR 2013)* (2013). 10

[BB12] Beck J., Burg K.: http://cinemagraphs.com. 10

[BBM*01] Buehler C., Bosse M., McMillan L., Gortler S., Cohen M.: Unstructured lumigraph rendering. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New

York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 425–432. 2, 21, 65, 114

[BCS97] Bregler C., Covell M., Slaney M.: Video rewrite: Driving visual speech with audio. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 353–360. 9

[BES17] Bochinski E., Eiselein V., Sikora T.: High-speed tracking-by-detection without using image information. In *Advanced Video and Signal Based Surveillance (AVSS), 2017 14th IEEE International Conference on* (2017), IEEE, pp. 1–6. 104, 116, 117, 118, 119, 125

[Bre01] Breiman L.: Random forests. *Machine learning 45*, 1 (2001), 5–32. 29, 33, 55

[BSHK04] Bhat K. S., Seitz S. M., Hodgins J. K., Khosla P. K.: Flow-based video synthesis and editing. In *ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), SIGGRAPH '04, ACM, pp. 360–363. 11, 12, 38

[BVZ99] Boykov Y., Veksler O., Zabih R.: Fast approximate energy minimization via graph cuts. In *ICCV* (1999), pp. 377–384. 9, 112, 113

[CAC*02] Chuang Y.-Y., Agarwala A., Curless B., Salesin D. H., Szeliski R.: Video matting of complex scenes. *ACM Transactions on Graphics 21*, 3 (July 2002), 243–248. Sepcial Issue of the SIGGRAPH 2002 Proceedings. 55

[Can86] Canny J.: A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on PAMI-8*, 6 (Nov 1986), 679–698. 54, 98

[CCM16] Chang C.-S., Chu H.-K., Mitra N. J.: Interactive videos: Plausible video editing using sparse structure points. *Computer Graphics Forum (Proc. Eurographics) 35* (2016). 19

[CDSHD13] Chaurasia G., Duchene S., Sorkine-Hornung O., Drettakis G.: Depth synthesis and local warps for plausible image-based navigation. *ACM Trans. Graph. 32*, 3 (July 2013), 30:1–30:12. 2, 21, 24

[CET01] Cootes T. F., Edwards G. J., Taylor C. J.: Active appearance models. *IEEE Transactions on pattern analysis and machine intelligence 23*, 6 (2001), 681–685. 124

[CK14] Chen Q., Koltun V.: Fast mrf optimization with application to depth reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 3914–3921. 83

[CLR11] Couture V., Langer M., Roy S.: Panoramic stereo video textures. In *Computer Vision (ICCV), 2011 IEEE International Conference on* (Nov 2011), pp. 1251–1258. 9, 10

[CPW*11] Chen J., Paris S., Wang J., Matusik W., Cohen M., Durand F.: The video mesh: A data structure for image-based three-dimensional video editing. In *Computational Photography (ICCP), 2011 IEEE International Conference on* (2011), IEEE, pp. 1–8. 13, 14

[CRZ00] Criminisi A., Reid I., Zisserman A.: Single view metrology. *International Journal of Computer Vision 40*, 2 (2000), 123–148. 98

[CVCH14] Casas D., Volino M., Collomosse J., Hilton A.: 4d video textures for interactive character appearance. *Computer Graphics Forum (Proc. Eurographics 2014) 33*, 2 (2014). 16, 18, 19, 129

[CZS*13] Chen T., Zhu Z., Shamir A., Hu S.-M., Cohen-Or D.: 3-sweep: extracting editable objects from a single photo. *ACM Transactions on Graphics (TOG) 32*, 6 (2013), 195. 17

REFERENCES

[DRB*08] Dragicevic P., Ramos G., Bibliowitcz J., Nowrouzezahrai D., Balakrishnan R., Singh K.: Video browsing by direct manipulation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2008), CHI '08, ACM, pp. 237–246. 14

[DT05] Dalal N., Triggs B.: Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on* (June 2005), vol. 1, pp. 886–893 vol. 1. 33

[EF15] Eigen D., Fergus R.: Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision* (2015), pp. 2650–2658. 18

[Far03] Farnebäck G.: Two-frame motion estimation based on polynomial expansion. In *Image Analysis*. Springer, 2003, pp. 363–370. 75

[FB81] Fischler M. A., Bolles R. C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM 24*, 6 (1981), 381–395. 20

[FG14] Fuhrmann S., Goesele M.: Floating scale surface reconstruction. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 46. 2, 3

[FH*15] Furukawa Y., Hernández C., et al.: Multi-view stereo: A tutorial. *Foundations and Trends® in Computer Graphics and Vision 9*, 1-2 (2015), 1–148. 20, 21, 24, 110, 111, 114

[FJS96] Finkelstein A., Jacobs C. E., Salesin D. H.: Multiresolution video. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 281–290. 9

[FNZ*09] Flagg M., Nakazawa A., Zhang Q., Kang S. B., Ryu Y. K., Essa I., Rehg J. M.: Human video textures. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2009), I3D '09, ACM, pp. 199–206. 16, 38

[GGC*08] Goldman D. B., Gonterman C., Curless B., Salesin D., Seitz S. M.: Video object annotation, navigation, and composition. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology* (New York, NY, USA, 2008), UIST '08, ACM, pp. 3–12. 13, 14

[GGSC96] Gortler S. J., Grzeszczuk R., Szeliski R., Cohen M. F.: The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), ACM, pp. 43–54. 21

[GHTC03] Gao X.-S., Hou X.-R., Tang J., Cheng H.-F.: Complete solution classification for the perspective-three-point problem. *IEEE Trans. Pattern Anal. Mach. Intell. 25*, 8 (Aug. 2003), 930–943. 102

[GLU12] Geiger A., Lenz P., Urtasun R.: Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)* (2012). 109

[HAA97] Horry Y., Anjyo K.-I., Arai K.: Tour into the picture: Using a spidery mesh interface to make animation from a single image. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 225–232. 17

[HASK17] Hedman P., Alsisan S., Szeliski R., Kopf J.: Casual 3D Photography. 234:1–234:15. 2, 21

[HEH05] Hoiem D., Efros A. A., Hebert M.: Automatic photo pop-up. In *ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), SIGGRAPH '05, ACM, pp. 577–584. 17

[Hit41] HITCHCOCK F. L.: The distribution of a product from several sources to numerous localities. 32

[HJO*01] HERTZMANN A., JACOBS C. E., OLIVER N., CURLESS B., SALESIN D. H.: Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM, pp. 327–340. 39, 72, 86

[HRDB16] HEDMAN P., RITSCHEL T., DRETTAKIS G., BROSTOW G.: Scalable inside-out image-based rendering. *ACM Trans. Graph. 35*, 6 (Nov. 2016), 231:1–231:11. 2, 21, 24

[JMD*12] JOSHI N., MEHTA S., DRUCKER S., STOLLNITZ E., HOPPE H., UYTTENDAELE M., COHEN M.: Cliplets: Juxtaposing still and dynamic imagery. UIST. 10, 11, 12, 22, 38, 75, 90, 91

[Joy12] JOYLABZ: http://makeymakey.com. 39, 72, 86

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 473–482. 15, 40, 47

[Kip14] KIPP M.: Anvil: A universal video research tool. In *Handbook of Corpus Phonology. Oxford University Press.* 2014, ch. 21, pp. 420–436. 77

[Kol06] KOLMOGOROV V.: Convergent tree-reweighted message passing for energy minimization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 28*, 10 (2006), 1568–1583. 83

[KSC*01] KLEIN A., SLOAN P., COLBURN A., FINKELSTEIN A., COHEN M. F.: Video cubism. *Technical Report MSR-TR-2001-45* (2001). 12

[KSE*03] KWATRA V., SCHÖDL A., ESSA I., TURK G., BOBICK A.: Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003 22*, 3 (July 2003), 277–286. 4, 9, 11, 13, 22, 29, 75

[KSES14] Kholgade N., Simon T., Efros A., Sheikh Y.: 3d object manipulation in a single photograph using stock 3d models. *ACM Transactions on Computer Graphics 33*, 4 (2014). 17

[LCR*02] Lee J., Chai J., Reitsma P. S. A., Hodgins J. K., Pollard N. S.: Interactive control of avatars animated with human motion data. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 491–500. 15, 40, 47

[LFH15] Liao J., Finch M., Hoppe H.: Fast computation of seamless video loops. *ACM Trans. Graph. 34*, 6 (Oct. 2015), 197:1–197:10. 10, 12

[LH96] Levoy M., Hanrahan P.: Light field rendering. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 31–42. 2

[LHB15] Lebeda K., Hadfield S., Bowden R.: Dense rigid reconstruction from unstructured discontinuous video. In *Proceedings of the ICCV workshop on 3D Representation and Recognition* (December 2015). 19, 130

[LJH13] Liao Z., Joshi N., Hoppe H.: Automated video looping with progressive dynamism. *ACM Trans. Graph. 32*, 4 (July 2013), 77:1–77:10. 4, 10, 11, 12, 22, 28, 73, 90, 91

[LLVT03] Lewiner T., Lopes H., Vieira A. W., Tavares G.: Efficient implementation of marching cubes' cases with topological guarantees. *Journal of graphics tools 8*, 2 (2003), 1–15. 111, 113

[Low04] Lowe D. G.: Distinctive image features from scale-invariant keypoints. *International journal of computer vision 60*, 2 (2004), 91–110. 48

[LSD15] Long J., Shelhamer E., Darrell T.: Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference*

on Computer Vision and Pattern Recognition (2015), pp. 3431–3440. 53

[LWA*12] Lang M., Wang O., Aydin T. O., Smolic A., Gross M. H.: Practical temporal consistency for image-based graphics applications. *ACM Trans. Graph. 31*, 4 (2012), 34–1. 55

[LWB*10] Lee Y., Wampler K., Bernstein G., Popović J., Popović Z.: Motion fields for interactive character locomotion. In *ACM SIGGRAPH Asia 2010 Papers* (New York, NY, USA, 2010), SIGGRAPH ASIA '10, ACM, pp. 138:1–138:8. 47, 48, 67

[LYGC15] Liao Z., Yu Y., Gong B., Cheng L.: Audeosynth: Music-driven video montage. *ACM Trans. Graph. 34*, 4 (July 2015), 68:1–68:10. 13

[LYT*08] Liu C., Yuen J., Torralba A., Sivic J., Freeman W. T.: Sift flow: Dense correspondence across different scenes. In *Computer Vision–ECCV 2008*. Springer, 2008, pp. 28–42. 10

[LZW*13] Lu S.-P., Zhang S.-H., Wei J., Hu S.-M., Martin R. R.: Timeline editing of objects in video. *IEEE Transactions on Visualization and Computer Graphics 19*, 7 (2013), 1218–1227. 12, 13, 14, 22, 70, 75, 90, 91

[MACKB14] Mac Aodha O., Campbell N. D., Kautz J., Brostow G. J.: Hierarchical Subquery Evaluation for Active Learning on a Graph. In *CVPR* (2014). 34, 41

[MAFK17] Mousavian A., Anguelov D., Flynn J., Kosecka J.: 3d bounding box estimation using deep learning and geometry. In *CVPR* (2017). 18

[Mey92] Meyer F.: Color image segmentation. In *Image Processing and its Applications, 1992., International Conference on* (1992), pp. 303–306. 54

[MLTR*16] Milan A., Leal-Taixé L., Reid I., Roth S., Schindler K.: MOT16: A benchmark for multi-object tracking. *arXiv:1603.00831 [cs]* (Mar. 2016). arXiv: 1603.00831. 104, 118

[MMMO] Moulon P., Monasse P., Marlet R., Others: Openmvg. an open multiple view geometry library. https://github.com/openMVG/openMVG. 20, 98

[NM65] Nelder J. A., Mead R.: A simplex method for function minimization. *The computer journal 7*, 4 (1965), 308–313. 100, 110

[NNL13] Nguyen C., Niu Y., Liu F.: Direct manipulation video navigation in 3d. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2013), CHI '13, ACM, pp. 1169–1172. 14

[NNL14] Nguyen C., Niu Y., Liu F.: Direct manipulation video navigation on touch screens. In *Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services* (2014), ACM, pp. 273–282. 14

[NP15] Nebehay G., Pflugfelder R.: Clustering of Static-Adaptive correspondences for deformable object tracking. In *Computer Vision and Pattern Recognition* (June 2015), IEEE. 74

[OCDD01] Oh B. M., Chen M., Dorsey J., Durand F.: Image-based modeling and photo editing. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 433–442. 17

[Ols12] Olsen D.: CHI 2012 Lifetime Achievement in Research Award. 68, 69, 91, 92

[OSGO12] Orghidan R., Salvi J., Gordan M., Orza B.: Camera calibration using two or three vanishing points. In *Computer Science and Information Systems (FedCSIS), 2012 Federated Conference on* (2012), IEEE, pp. 123–130. 122

[PD07] PERRONNIN F., DANCE C.: Fisher kernels on visual vocabularies for image categorization. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on* (June 2007), pp. 1–8. 33

[Pet16] PETERS M.: The Birth of Loop. 69

[PGB03] PÉREZ P., GANGNET M., BLAKE A.: Poisson image editing. In *ACM Transactions on Graphics (TOG)* (2003), vol. 22, ACM, pp. 313–318. 71, 74, 84

[Pic04] PICCARDI M.: Background subtraction techniques: a review. In *Systems, man and cybernetics, 2004 IEEE international conference on* (2004), vol. 4, IEEE, pp. 3099–3104. 53

[PKC*16] PRADA F., KAZHDAN M., CHUANG M., COLLET A., HOPPE H.: Motion graphs for unstructured textured meshes. *ACM Trans. Graph. 35*, 4 (July 2016), 108:1–108:14. 129

[PPHL98] POLLARD S., PILU M., HAYES S., LORUSSO A.: View synthesis by trinocular edge matching and transfer. In *Proceedings of the 4th IEEE Workshop on Applications of Computer Vision (WACV'98)* (Washington, DC, USA, 1998), WACV '98, IEEE Computer Society, pp. 168–. 9

[PRAP08] PRITCH Y., RAV-ACHA A., PELEG S.: Nonchronological video synopsis and indexing. *IEEE Transactions on Pattern Analysis and Machine Intelligence 30*, 11 (2008), 1971–1984. 12

[Pri12] PRINCE S. J.: *Computer vision: models, learning, and inference.* Cambridge University Press, 2012. 44

[PVGV*04] POLLEFEYS M., VAN GOOL L., VERGAUWEN M., VERBIEST F., CORNELIS K., TOPS J., KOCH R.: Visual modeling with a hand-held camera. *International Journal of Computer Vision 59*, 3 (2004), 207–232. 98

[RAPLP05] Rav-Acha A., Pritch Y., Lischinski D., Peleg S.: Evolving time fronts: Spatio-temporal video warping. In *SIGGRAPH'05* (2005). 13

[RNR*17] Rematas K., Nguyen C. H., Ritschel T., Fritz M., Tuytelaars T.: Novel views of objects from a single image. *IEEE Transactions on Pattern Analysis and Machine Intelligence 39*, 8 (Aug 2017), 1576–1590. 17

[RTG98] Rubner Y., Tomasi C., Guibas L. J.: A metric for distributions with applications to image databases. In *Proceedings of the Sixth International Conference on Computer Vision* (Washington, DC, USA, 1998), ICCV '98, IEEE Computer Society, pp. 59–. 31, 32

[RWSG13] Rüegg J., Wang O., Smolic A., Gross M.: Ducttake: Spatiotemporal video compositing. In *Computer Graphics Forum* (2013), vol. 32, Wiley Online Library, pp. 51–61. 13

[SCK08] Shum H.-Y., Chan S.-C., Kang S. B.: *Image-based rendering*. Springer Science & Business Media, 2008. 20, 21, 114

[SE01] Schödl A., Essa I. A.: Machine learning for video-based rendering. In *Advances in Neural Information Processing Systems 13*, Leen T., Dietterich T., Tresp V., (Eds.). MIT Press, 2001, pp. 1002–1008. 15

[SE02] Schödl A., Essa I. A.: Controlled animation of video sprites. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2002), SCA '02, ACM, pp. 121–127. 15, 16, 38, 47

[SF16] Schönberger J. L., Frahm J.-M.: Structure-from-motion revisited. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016). 20, 98

[SLWSS15] Sevilla-Lara L., Wulff J., Sunkavalli K., Shechtman E.: Smooth loops from unconstrained video. In *Proceedings of the 26th Eurographics Symposium on Rendering* (Aire-la-Ville, Switzerland,

Switzerland, 2015), EGSR '15, Eurographics Association, pp. 99–107. 4, 10, 11, 28, 48, 94

[SN13] Shah R., Narayanan P.: Interactive video manipulation using object trajectories and scene backgrounds. *Circuits and Systems for Video Technology, IEEE Transactions on 23*, 9 (Sept 2013), 1565–1576. 13

[SSS08] Snavely N., Seitz S. M., Szeliski R.: Modeling the world from Internet photo collections. *International Journal of Computer Vision 80*, 2 (November 2008), 189–210. 20, 98

[SSSE00] Schödl A., Szeliski R., Salesin D. H., Essa I.: Video textures. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., pp. 489–498. 3, 4, 8, 10, 11, 12, 14, 15, 16, 22, 27, 28, 29, 31, 37, 38, 39, 40, 42, 49, 58, 59, 60, 61, 72, 84, 127

[ST06] Sand P., Teller S.: Particle video: Long-range motion estimation using point trajectories. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)* (2006), vol. 2, pp. 2195–2202. 14

[Tar09] Tardif J.-P.: Non-iterative approach for fast and accurate vanishing point detection. In *Computer Vision, 2009 IEEE 12th International Conference on* (2009), IEEE, pp. 1250–1257. 98, 99, 100

[TDB15] Tatarchenko M., Dosovitskiy A., Brox T.: Single-view to multi-view: Reconstructing unseen views with a convolutional network. *arXiv preprint arXiv:1511.06702* (2015). 18

[TPSK11] Tompkin J., Pece F., Subr K., Kautz J.: Towards moment images: Automatic cinemagraphs. In *Visual Media Production (CVMP), 2011 Conference for* (November 2011), pp. 87–93. 10, 11, 12, 22, 29

[vdHDT*07] VAN DEN HENGEL A., DICK A., THORMÄHLEN T., WARD B., TORR P. H. S.: Videotrace: Rapid interactive scene modelling from video. In *ACM SIGGRAPH 2007 Papers* (New York, NY, USA, 2007), SIGGRAPH '07, ACM. 19

[vGJMR12] VON GIOI R. G., JAKUBOWICZ J., MOREL J.-M., RANDALL G.: Lsd: a line segment detector. *Image Processing On Line 2* (2012), 35–55. 99

[Wat89] WATKINS C. J. C. H.: *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989. 50, 60

[WBSS04] WANG Z., BOVIK A. C., SHEIKH H. R., SIMONCELLI E. P.: Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on 13*, 4 (2004), 600–612. 28

[WDC*15] WEN L., DU D., CAI Z., LEI Z., CHANG M., QI H., LIM J., YANG M., LYU S.: DETRAC: A new benchmark and protocol for multi-object detection and tracking. *arXiv CoRR abs/1511.04136* (2015). 104

[WLW*17] WANG L., LU Y., WANG H., ZHENG Y., YE H., XUE X.: Evolving boxes for fast vehicle detection. In *IEEE International Conference on Multimedia and Expo (ICME)* (2017), pp. 1135–1140. 104, 116, 118

[Wol90] WOLBERG G.: *Digital image warping*, vol. 10662. IEEE computer society press Los Alamitos, CA, 1990. 44

[WRB11] WEINLAND D., RONFARD R., BOYER E.: A survey of vision-based methods for action representation, segmentation and recognition. *Computer vision and image understanding 115*, 2 (2011), 224–241. 75

[WZJ16] WORKMAN S., ZHAI M., JACOBS N.: Horizon lines in the wild. In *British Machine Vision Conference (BMVC)* (2016). 99, 122

[XJXC12] Xiong C., Johnson D., Xu R., Corso J. J.: Random forests for metric learning with implicit pairwise position dependence. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2012), KDD '12, ACM, pp. 958–966. 33

[ZCC*12] Zheng Y., Chen X., Cheng M.-M., Zhou K., Hu S.-M., Mitra N. J.: Interactive images: Cuboid proxies for smart image manipulation. *ACM Transactions on Graphics 31*, 4 (2012), 99:1–99:11. 17

[ZGL*03] Zhu X., Ghahramani Z., Lafferty J., et al.: Semi-supervised learning using gaussian fields and harmonic functions. In *ICML* (2003), vol. 3, pp. 912–919. 36, 41, 42, 76, 77, 78

[ZTS*16] Zhou T., Tulsiani S., Sun W., Malik J., Efros A. A.: View synthesis by appearance flow. *CoRR abs/1605.03557* (2016). 18

[ZWJ16] Zhai M., Workman S., Jacobs N.: Detecting vanishing points using global image context in a non-manhattan world. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 5657–5665. 99, 100